FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework for Offline Wireless Network Experimentation

João Bernardo Martins de Sousa e Silva Mota

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Manuel Pereira Ricardo (PhD)

Second Supervisor: Helder Martins Fontes (MSc)

July 23, 2018

Framework for Offline Wireless Network Experimentation

João Bernardo Martins de Sousa e Silva Mota

Mestrado Integrado em Engenharia Informática e Computação

July 23, 2018

Abstract

Projects related to communications systems often, if not always, require testing in a real testbed. This involves setting up a mostly temporary experiment, which may require the use of several resources. If the results from the real world experiments turn out to be different from those obtained from simulations, the project will need to be re-evaluated in a simulation environment before repeating the same, costly, real world experiments. This cycle of testing and validation can repeat itself an arbitrary number of times before a suitable solution is found, which in turn can amount to a considerable investment in terms of resources and time.

This dissertation's project aims to minimize or even fully mitigate the amount of real world experimentation required to develop robust and reliable solutions by providing a system that allows its users to automatically create simulations whose conditions come very close to the ones found in the real world. Furthermore, the solution proposed in this dissertation would reduce the user's workload as well as minimize the possibility of human error in the process, by automating most of the tasks involved. This is achieved by storing layer one characteristics from previous real experiments, such as Signal-to-Noise Ratio (SNR) and node positions, and reproducing them in simulations instead of relying on the available Propagation Loss and Mobility models. By using this system, one can get more accurate results from a simulator, which means that when a project is finally tested in a real world environment, the probability that the obtained results differ significantly from those obtained in simulated environments is considerably lower, thus reducing the number of times the project needs to be tested in real testbeds. The solution proposed in this dissertation can also become a platform that enables wide and open access to network researchers data. This would simplify sharing of data and is useful in many cases, such as the validation of results published in scientific papers and those obtained from network projects in general.

To validate the solution proposed in this document, a proof-of-concept version of the system was developed. The system is composed by three main components: 1) an information retrieval agent to be installed in the real testbed nodes; 2) a central storage system to store the layer one characteristics from real experiments; and 3) a web-based user interface to browse data from past experiments and download automatically generated trace-based simulations. The system was subjected to real usage, where its purpose was fulfilled by successfully recording a real laboratory testbed experiment and reproducing it as a trace-based simulation. Furthermore, a questionnaire was distributed among a group of researchers, the results of which showed a significant acceptance of the framework as an improvement upon the current methodology.

Resumo

É com frequência que os projetos relacionados com redes de comunicações exigem que se façam testes em ambientes reais. Para efetuar estes testes é necessário montar um cenário, muitas vezes temporário, que consumirá uma quantidade significativa de recursos. Se se vier a verificar que os resultados obtidos da experiência em ambiente real diferem dos resultados obtidos em ambientes de simulação num grau que ultrapasse o aceitável, o projeto terá que ser reavaliado em ambientes de simulação e, posteriormente, voltar a ser testado em ambientes reais. Este processo de teste e validação poderá repetir-se um número arbitrário de vezes até que se consigam resultados aceitáveis, o que resulta num elevado consumo de recursos.

Com esta dissertação, pretende-se mitigar ou mesmo eliminar completamente a necessidade de repetir experiências em ambientes reais para o desenvolvimento de soluções robustas e de confiança, fornecendo aos utilizadores um sistema que permite criar automaticamente simulações cujas condições se aproximam muito daquilo que é encontrado no mundo real. Além disso, a solução proposta nesta dissertação tem como objetivo eliminar a quantidade de trabalho que o utilizador teria normalmente, assim como a possibilidade de erro humano no processo, através da automatização da maioria dos passos envolvidos. Isto é conseguido através do armazenamento de características da camada física da rede, tais como Signal-to-Noise Ratio (SNR) e as posições dos nós de experiências feitas anteriormente, substituindo-as posteriomente pelos modelos relevantes, nomeadamente, o PropagationLossModel e o MobilityModel. A utilização deste sistema permite a obtenção de resultados de simulações mais próximos dos esperados em ambientes reais, o que diminui significativamente a probabilidade de os resultados obtidos em ambos os casos diferirem significativamente, por sua vez diminuindo a necessidade de voltar atrás no ciclo de desenvolvimento e poupando assim os recursos que teriam sido utilizados ao longo do processo. Esta solução tem ainda o potencial de se tornar numa plataforma que permite o acesso generalizado a dados de investigadores da área de redes de comunicação. Isto simplificaria a partilha de dados e tem vários casos de uso, tal como a validação de resultados publicados em artigos científicos ou em projetos de comunicações no geral.

Como validação da solução proposta neste documento, foi desenvolvido um sistema que consitui uma prova de conceito da mesma. Esse sistema é composto por 3 componentes principais: 1) um agente de recolha de informação; 2) am sistema de armazenamento central e; 3) uma interface web para os utilizadores. O sistema foi submetido a um cenário de experimentação real onde conseguiu cumprir as suas tarefas com sucesso, registando os dados da experiência e permitindo a sua reprodução como uma simulação *trace-based*. Além disso, foi distribuído um questionário a um grupo de investigadores, cujos resultados mostraram uma clara aceitação da *framework* como uma melhoria sobre a metodologia atualmente usada.

Acknowledgements

The culmination of this dissertation would not have been possible without assistance, in one form or another, from everyone in my life. I dedicate this note and this project to all of you.

First and foremost, I would like to thank my supervisors on this project: Helder Fontes and Prof. Manuel Ricardo. Without their tireless dedication and unparalleled support, this project would have been lost.

Thank you also to everyone at the Wireless Networks (WiN) research team for contributing with their opinions on the project. This project was developed for them and their feedback gave me hope that it would have the positive impact that I wanted it to have on the scientific community and the field of network research.

A special mention of thanks goes to my family for providing me with the means and the opportunity to pursue my aspirations to the fullest extent of my ability. This dissertation is nothing if not the product of the trust and hopes they put on me.

Finally, I would like to thank and also apologize to all of my friends for putting up with many a get-together in which I could not be parted from my computer. Thank you also for being there to provide catharsis, encouragement, advice and, most of all, for letting me grow with and alongside you.

João Bernardo Mota

"You must never give in to despair. Allow yourself to slip down that road and you surrender to your lowest instincts.
In the darkest times, hope is something you give yourself. That is the meaning of inner strength".

Iroh

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BASE	Basically Available, Soft State, Eventually Consistent
BSON	Binary JSON
CAP	Consistency, Availability, Partition Tolerance
CONCRETE	Control and Classify Repeatable Testbed Experiments
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update, Delete
CPU	Central Processing Unit
CQL	Cassandra Query Language
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
ES5	ECMAScript 5
GPS	Global Positioning System
GUI	Graphical User Interface
HDD	Hard Drive Disk
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
LAMP	Linux Apache MySQL PHP
LTE	Long Term Evolution
MEAN	MongoDB, ExpressJS, AngularJS and NodeJS
MVC	Model-View-Controller
MVCC	Multiversion Concurrency Control
MVVM	Model-View-ViewModel
MVW	Model-View-Whatever
NoSQL	Not only SQL
ORM	Object-Relational Mapping
RAM	Random Access Memory
REST	Representational State Transfer
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SDN	Software Defined Networks
SDR	Software Defined Radio
SNR	Signal to Noise Ratio

SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UDP	User Datagram Protocol
URL	Universal Resource Locator
USB	Universal Serial Bus
XML	Extensible Markup Language
WLAN	Wireless Local Area Network
WWW	World Wide Web

Chapter 1

Introduction

1.1 Context

The life-cycle of a software development project can follow many models. Each model has its own advantages and disadvantages as well as its own use cases. In the case of projects related to communications systems, the commonly used model is akin to the classic waterfall model, which is described in Figure 1.1.

The first three phases of this model concern themselves with the analysis of the system requirements and the design and implementation of the system. The fourth phase, which involves testing the developed solutions, is the focus of this dissertation and where it has a greater impact. Generally, in communications systems related projects, this fourth phase can be divided into two sub-phases: testing in simulation and testing in real testbeds.

Usually, a project is first tested and tweaked in a simulated environment until satisfactory results are obtained. Network simulation tools are a considerable asset in this field, since they allow for an almost complete manipulation of the environment's variables with virtually no expense resource-wise, while also obtaining repeatable and reproducible results. This means that, in simulations, a wide range of test conditions can be created to assess the behaviour of the solution under study. From those results, the project can either advance to real environment testing or retreat into development or design phase in order to make the necessary tweaks so that results in the simulation become satisfactory.

When the solution is ready to be tested in a real environment, preparations have to be made regarding the setup and execution of the experiment:

- Equipment needs to be acquired or reserved for the experiment;
- Permits may need to be obtained depending on the nature of the experiment;
- Venues may have to be bought, leased or reserved for the purposes of the experiment;

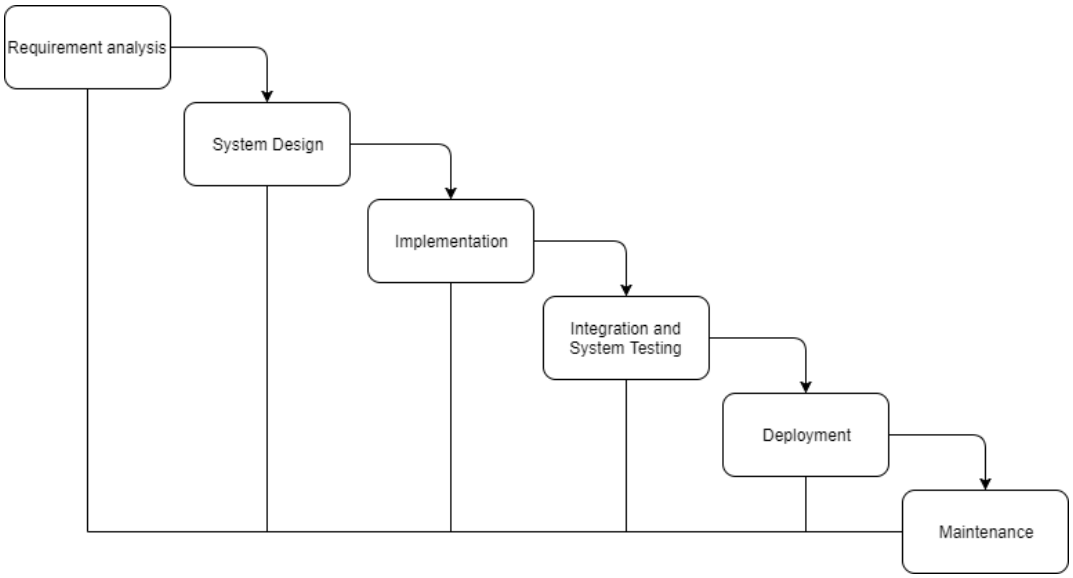

Figure 1.1: Classic waterfall model diagram.

- One or more persons may need to be present to supervise the experiment or execute tasks that need to be done manually or require some sort of human interaction.

All of this can consume a significant amount of resources. If the results of the experiment turn out to diverge from those obtained in simulation past the limit of acceptability, the project has to go back to the implementation or design phases for re-evaluation.

This usually happens because simulated environments can never perfectly reproduce the conditions encountered in the real world, since many naturally occurring phenomena are either poorly understood in the light of today's scientific knowledge, or they are understood, but they are too complex to reproduce in a timely manner given the processing power which is currently available. Instead, simulators resort to approximate models of these phenomena which are relatively accurate and much faster to process. These models perform acceptably in cases in which the setup being simulated is tested in relatively stable conditions. They tend to perform poorly, however, when the setup is placed under edge conditions. The inadequacy of traditional simulation methods is especially apparent when simulating projects related to aerial or sub-aquatic networks, where such edge conditions are constant and highly dynamic in nature.

A number of approaches are proposed in the state of the art to deal with the inaccuracy of simulation models, one of which is the trace-based network simulation. Trace-based network simulation consists in replacing simulation models by characteristics extracted from previous real experiments. By doing this, not only can results obtained from simulations become more akin to what one would expect in experiments with real testbeds, but the processing power necessary to run a simulation is also significantly reduced, since the calculations required by simulation models are no longer necessary. An issue with this approach, however, is the overhead generated by all the steps necessary to capture, process, organize and reuse the necessary data without incurring in human errors.

1.2 Motivations and Goals

This dissertation arose from the need to expedite and automate the process of data collection, processing and storage for the purpose of its reuse in trace-based simulation. The main goal is to design and implement a prototype of a framework to automate the process of extracting, processing, storing and reusing data from previous experiments with the goal of minimizing the user's workload and the number of user interactions necessary to get from an experiment's results to the reproduction of that experiment in a simulated environment. A secondary goal of this framework is to provide an open platform which facilitates access to experiment data from multiple users. This would result in a new-found abundance of test data for future communications systems related projects as well as a significant simplification of the process of verifying results reported in scientific papers and networking projects in general. This is achieved by creating a platform for users to access data and download pre-configured trace-based simulations.

The proposed system would be comprised by a generic data collection agent to be installed in network devices and a central storage system, consisting of a central database, a web service and a web application to act as its interface to the users, which would be available on the Internet.

This system should also conform to the following requirements:

- Robustness: The system should be able to handle unforeseen issues, such as connection losses and insufficient memory gracefully;
- Lightness: The part of the system that runs directly on the network nodes should consume few enough network and processing resources that the outcome of the experiment is not affected;
- Security: There should be no data leaks, either of user information or experiment data in the system. To this end, all data transmissions must be encrypted and data should remain private unless system administrators choose to make it public.

The project's usefulness will be validated by a comparison between users attempting to capture data to reuse in future trace-based simulations manually, as was done until now, and users using the framework proposed in this dissertation. Concrete metrics for this include the amount of time spent and the number of user interactions.

1.3 Document Structure

The remainder of this document is structured as follows: Chapter 2 contains the literature review for this dissertation as well as the technological study and comparison that justified the technology choices for the framework implementation. Chapter 3 introduces the problems and requirements regarding the reproduction of real experiments in simulation, as well as a high-level architectural description of the proposed solution. Chapter 4 reveals some lower-level implementation details as well as the technology choices made for the framework's implementation. Chapter 5 presents

the methods through which this dissertation's proposed solution was validated as well as the corresponding results. Finally, Chapter 6 consists of an overview of the main conclusions of each chapter, followed by the enumeration of this dissertation's main contributions as well as all future work suggestions.

Chapter 2

Literature Review

This chapter contains an analysis of the state-of-the-art concerning network experimentation frameworks, trace-based network simulation approaches, data storage techniques and technologies, and web application frameworks and technologies.

This chapter also presents a comparison of various web frameworks and database management systems that helps justify the choice of technologies for the implementation component of this dissertation.

2.1 Network experimentation frameworks

Nowadays, there are a number of frameworks for network experimentation. These frameworks offer online services which provide access to different real-world testbeds. In these testbeds, an experimenter can select a number of mobile and stationary nodes according to his specification. Once reserved, the nodes can be configured automatically or manually to run the necessary experiments. These frameworks can also provide full performance reports after the experiment is finished.

Geni [Gena] has one such framework, with testbeds spread all throughout north America, as seen in Figure 2.1 and also three testbeds in Belgium. Testbeds provided through Geni allow users to specify traffic handling protocols in the provided switches and routers and the installation and configuration of the users' own protocols above layer 2.

Fed4FIRE+ [Feda] is an European project under the supervision of the European Union with the goal of providing infrastructure for network research and development. Fed4Fire+'s framework is very similar to Geni's. In fact, they are interoperable in some types of experiments. The project currently offers a number of testbeds, seen in Figure 2.2, to network developers, targeting different communities within the field of network research. Experimentation using these testbeds can be done via request to the regulating entity, which will schedule a time for the experiment to be run.

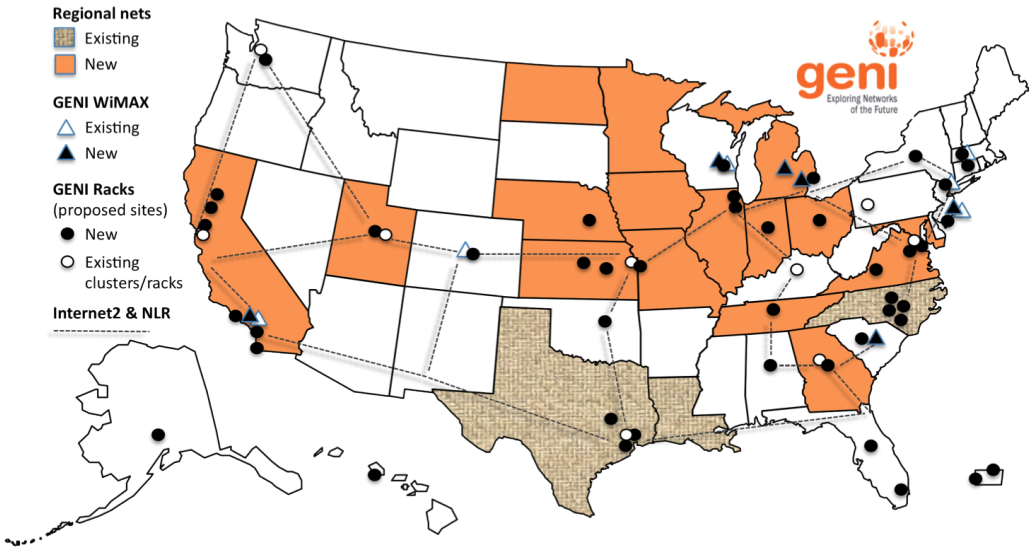

Figure 2.1: Testbeds available through Geni and their geographical locations [genb].

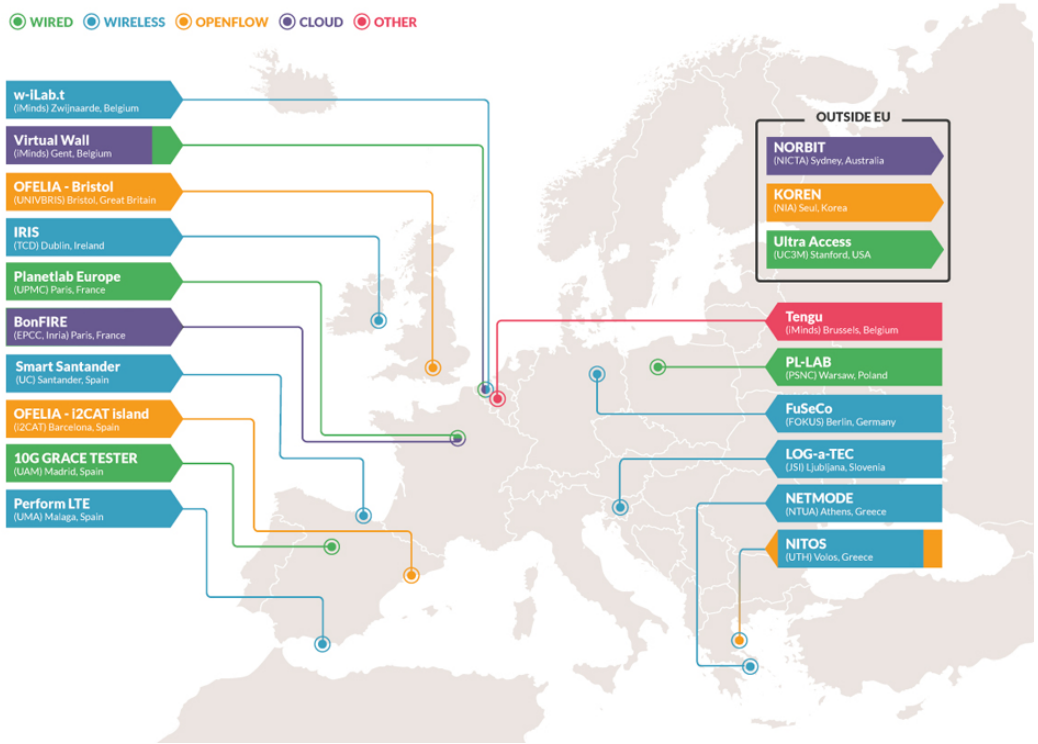

Figure 2.2: Testbeds available through Fed4FIRE+ and their geographical locations [fedb].

From all the testbeds federated by Geni and Fed4FIRE+, the ones most relevant to this project are w-iLab.t and NITOS. w-iLab.t is a testbed for Wi-Fi and sensor network experimentation located in Ghent, Belgium. This testbed provides a 60x22.5 meter facility containing 80 nodes: 60 stationary and 20 mobile nodes. This testbed, despite its highly configurable nature and seemingly stable and isolated environment, can produce different results for the same experiment, rendering it non-repeatable and irreproducible. This may be caused by external phenomena, such as the noise floor and interference, that could affect the performance results.

NITOS is designed to achieve repeatability and reproducibility in the experiments carried out within its testbeds. Located in Volos, Greece, it consists of two testbeds: one outdoor testbed with heterogeneous nodes supporting Wi-Fi, WiMax and Long Term Evolution (LTE); and an indoor testbed with powerful nodes supporting a Wi-Fi and Bluetooth. NITOS also provides Software Defined Networking (SDN) and Software Defined Radio (SDR) capabilities in its testbeds, making it easier for a developer to set up a custom scenario.

These testbeds try to achieve repeatability and reproducibility in experiments through the CONtrol and Classify REpeatable Testbed Experiments (CONCRETE) tool. CONCRETE [KLM+12] is a tool that tries to ensure repeatability and reproducibility of experimentation results. It does so by scheduling multiple runs of the same experiment, trying to establish a correlation between their results and discarding runs with results that are deemed as unstable, considering only the most stable runs as representations of the system's operation. This produces an apparent improvement in the experiment's repeatability and reproducibility. The problem with this approach is that the unstable runs that are discarded are also possible states for the system and can be a sign of circumstances unforeseen by the developers of the solution. In conclusion, CONCRETE can only ensure the reproducibility of stable experiments, and thus this tool is unsuited to the extreme and highly unstable scenarios of emerging scenarios, such as flying and underwater networks.

2.2 Trace-based network simulation

As stated in Section 1.1, standard simulations may oftentimes be inadequate for the reproduction of past experimentation results, as shown by Khan et al. [KAN+13]. Trace-based network simulation appeared in the state-of-the-art as an answer to bridge the gap between simulation and experimentation results, especially for cases where simulation models are unable to cope with the complex phenomena that affect the system's operation in a real environment.

In the state-of-the-art, there are three different approaches to trace-based network simulation:

- Packet-based Replay Systems;
- Application Layer Replay Systems;
- Physical Layer Replay Systems.

Owerzarski [OL04] introduces the concept of packet-based replay in network simulators by building a replay module in the NS simulator that takes files containing flow information such

as time stamps and number and size of packets. This information can be extracted from any packet-level network information capture provided by the classical programs such as TCPdump or wireshark.

Application layer replay systems appeared to bridge the gap between the behaviour of real application layer protocols and the simulation models that are available within simulators such as ns-3. Agrawal [AV16] proposes a trace-based application layer simulator that, given a trace extracted from a user session where the designated application is running, can generate traffic that is faithful to real application traffic behaviour. The proposed model controls application layer delays such as user think times and lets the simulator take care of controlling the remainder of layers' behaviours. This model is also generic enough that it can generate traffic for a large variety application layer protocols.

A Physical layer replay system, contrary to the approaches proposed above consists in extracting information about the behaviour of the physical layer and letting the simulator control the behaviour of upper layers. A particular characteristic of this approach is that the behaviour of the upper layers can be modified so that a solution can be continuously improved even without the need to experiment in a real testbed. Fontes [FCR17] proposes a model to implement this kind of trace-based simulation in ns-3. This model significantly increases the similarity of simulation results to the ones obtained in real testbeds, enabling perpetuation of experiments in a simulated environment. This model also makes simulations faster since calculations for loss model values are no longer needed, as they are now imposed by the given trace. The model has recently been improved to accept Multiple Access scenarios [FCR18].

2.3 Technology Comparison

This section presents an introduction to the technologies that will be necessary for the project's accomplishment, along with examples of some implementations with a description of their advantages and disadvantages.

2.3.1 Database Technologies

A crucial part of this project is the storage necessary to hold the experimental results. Databases can be used to properly organize the data and, later, efficiently access it. To that effect, database technologies are explored in this section.

Database technologies can currently be divided into two big categories: Relational databases and NoSQL databases.

Relational databases were the norm for decades and are still a non-negligible part of all databases in use today. These kinds of databases are mainly characterized by their Structured Query Language (SQL) and ACID (Atomicity, Consistency, Integrity, Durability) compliance, making them particularly suited for use cases where the data model is constant, complex queries are required and the data needs to be consistent at all times. The most pressing examples of this are bank databases, where potentially thousands, if not millions, of transactions occur per second

and balance values need to stay consistent so that nothing is lost (or gained) due to transactional errors.

As stated by a number of authors [Lea10, Ore10, CDG+08], recent technological improvements have made it feasible to store and massive quantities of data. Faced with unusually large and complex data sets, problems started to appear that relational databases were not suited to solve, such as non-solid schemas and scalability issues. Thus, different database technologies surfaced, all uniting under the name NoSQL, meaning "Not Only SQL".

NoSQL databases can't be fully categorized into a set of features, since, unlike SQL, which is a tried and tested norm, NoSQL is a term used to describe all technologies which are not SQL-compliant. That said, there are a number of ways to differentiate NoSQL databases, such as the type of data model and storage they offer [Cat11, SA12, Lea10], their standing on the ACID-BASE spectrum [Str11], and where they fall according to the CAP theorem [Str11, Bre00].

In terms of data models, NoSQL databases can be divided into four categories: key-value stores, document stores, column stores and graph-based.

ACID, as described by [HR83] is a set of properties that relational databases ensure for transactions. Transactions should be Atomic, Consistent, Isolated and Durable. Atomicity means that all changes concerning one row or set of rows should all be processed or all fail together. This ensures that crashes do not leave the database in an inconsistent state. Consistency means that a transaction transitions the database from one consistent state to another. In other words, transactions should not alter the data in such a way that constraints or other existing checks are violated. Isolation means that concurrent changes to the database behave as sequential changes. This is necessary to ensure that the most up to date values are used during changes, so that consistency can be maintained. Durability means that a change persists in the database, even through issues such as crashes, power losses or network losses. Ensuring these properties leads to a mostly full proof, valid, transaction model.

BASE, as described by Pritchett [Pri08] lies on the other side of the spectrum, meaning that data should be Basically Available, Soft state and Eventually consistent. Most NoSQL technologies inherently lean more toward this set of properties. Basically available means that all data in the database is available insofar as none of the nodes crash. Crashing nodes may take time to recover and lead to potential data losses. Replication algorithms may take time to ensure that all changes are present in all nodes, thus the eventual consistency. Soft state means that the database's state may change even without user interaction. This is once again due to the replication algorithms performing updates across other nodes.

The CAP theorem [Bre00] can be represented by a Venn diagram as can be seen in Figure 2.3, taken from Lourenço's work [LCC+15]. What the theorem states is that database technologies can only ever ensure two of the three properties: Consistency, Atomicity or Partition Tolerance. The figure further shows some database technologies that implement certain options of the CAP theorem.

In this project, the central storage component should be able to indefinitely store what is essentially time series data for several experiments simultaneously. Furthermore, since the metrics

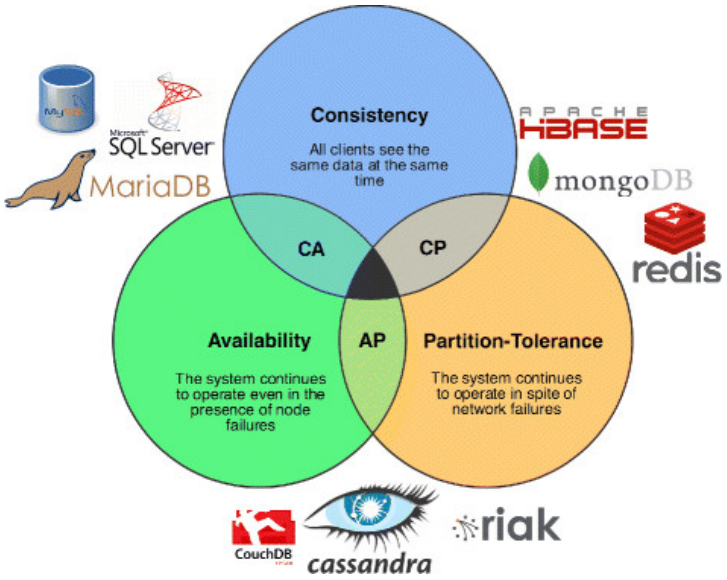

Figure 2.3: Visual representation of the CAP theorem [LCC+15].

to retrieve from network devices should be defined by the user, the need arises for a schema-less database. That said, the ideal database technology for this project would be one with good write performance, good scalability, and support for unstructured or semi-structured data. Also, since there will be no updates on any objects once they are stored, only insertions and deletions, transactional consistency is of no concern as far as this project goes. Relational databases can offer some of these features but NoSQL databases should provide better performance and scalability as well as support for semi-structured data. Therefore, the choice was made to use a NoSQL database.

Table 2.1 shows the characteristics of some of the currently most used NoSQL databases, namely MongoDB, Cassandra, CouchBase, HBase, and Aerospike. As can be seen, there are significant differences between all of these databases, so a further analysis of each technology is offered below.

MongoDB [Mon] is a document store database in which documents are encoded in BSON (binary JSON). It uses a master-slave model, meaning that writes are only processed on one node, which will eventually propagate its changes to the remaining nodes via a transaction log. Reads can be done from any node, which can result in inconsistencies in the data. For this to happen, however, the database would need to be under significant read/write loads. Consistency configurations in MongoDB allow the user to define the confirmation threshold for nodes to execute reads, making the database more or less consistent accordingly. The popularity of this database has lead to a number of interface APIs in different languages and for different programs, making MongoDB a relatively easy to integrate database.

Cassandra [Cas] on the other hand is a column store database with a shared-nothing architecture, which means that nodes are independent and there is no redundant data. Columns in Cassandra consist of the value, a timestamp used for consistency assurances and the name of the column. This database uses a SQL-like querying language called CQL, which smoothes the learn-

Table 2.1: Characteristic comparison of some NoSQL databases [LCC+15, Gos].

	MongoDB	*Cassandra*	*CouchBase*	*HBase*	*Aerospike*
Data model	Document store	Column store	Document store	Column store	Key-value store
CAP	CP	AP/CP	AP/CP	CP	AP
Consistency	Configurable	Configurable	Strong/Eventual	Configurable	Configurable
Durability	Configurable	Configurable	Configurable	Configurable	Configurable
Querying	Internal API	Internal API	Internal API	Internal API	Internal API
Concurrency	Master-slave	MVCC	MVCC	Optimistic	Read-committed

ing curve and eases the issuing of complex queries. Cassandra achieves highly tunable consistency by allowing users to specify the desired level of trade-off between consistency and latency. The database combines disk persistence with in-memory cache in order to achieve high write throughputs. The master-master architecture also ensures that performance is not as affected by high operation loads.

CouchBase [Cou] appears as the combination of CouchDB and MemBase, two other NoSQL databases. It is considered a document store database, but it can be used in key-value fashion. CouchBase ensures strong consistency within nodes but only eventual consistency throughout a cluster. Documents in this database are stored in what is called data buckets. Queries can be made through MapReduce interfaces in Javascript. CouchBase is meant to run in-memory so that it can hold the whole data set in RAM.

HBase [HBa] is meant as an open source version of Google's BigTable, and as such has similar capabilities. HBase is almost ACID compliant: atomicity is guaranteed at row-level; there are assurances that rows only move forward in time and that no rows result of interleaving operations; isolation exists at a read-committed level; and durability is ensured in the sense that data is written on the disk. Data stored in HBase is a binary array; there are no data types. Furthermore, this database does not support secondary indexing, meaning that data can only be found via its primary key or table scans. Communication with the database can be done via Java or REST APIs, or the Avro [avr] and Thrift [thr] protocols.

Aerospike [Aer] is a shared-nothing key-value database. It provides mainly Availability and Partition Tolerance, though it can also provide high consistency by immediately indexing data. This immediate indexation can be relaxed in order to gain availability. This database has a highly configurable failover, offering availability in Consistency-Partition Tolerance (CP) mode or high consistency in Availability-Partition Tolerance (AP) mode. Replication is achieved by concurrent writes to other nodes.

Regarding performance, Figure 2.4, taken directly from Lourenço [LCC+15] – who exhaustively enumerates and analyses database performance comparison papers – shows the strengths and weaknesses of the databases described above.

For this project, the chosen database technology should be able to endure big write bursts as well as be able to stay performant as the amount of stored data increases. It should be robust enough that it resists potential sporadic errors and durable enough that even if any crashes or loss

	Aerospike	Cassandra	Couchbase	CouchDB	HBase	MongoDB	Voldemort
Availability	⊞	⊞	⊞	⊞	−	−	⊞
Consistency	⊞	⊞	+	+	□	⊞	+
Durability	−	+	+	−	+	+	+
Maintainability	+	□	+	+	−	□	−
Read-Performance	+	−	⊞	□	−	⊞	+
Recovery Time	⊞	⊖	+	?	?	⊞	?
Reliability	−	+	−	+	+	⊞	?
Robustness	+	+	□	□	⊖	□	?
Scalability	⊞	⊞	⊞	−	⊞	−	+
Stabilization Time	⊖	+	+	?	?	⊖	?
Write-Performance	+	⊞	+	−	+	−	⊞

Legend:
⊞ Great
\+ Good
□ Average
− Mediocre
⊖ Bad
? Unknown/N.A.

Figure 2.4: Database performance comparison regarding a number of different features [LCC+15].

of power occurs, data will not be lost. Thus, some of the database technologies analysed above can already be excluded, namely: MongoDB and CouchDB, for their relatively poor write performance; and Aerospike, for its relatively poor durability assurances. On the other hand, the more suitable database technologies introduced above are: Cassandra, for being an all-around good database, but especially for its high write performance and ease of scalability; and CouchBase, for the high scalability capacity and high read-performance.

2.3.2 Web Services

The component that will link both router devices and users to the permanent storage database will be a web service. Web services bring many advantages as a form of communication, such as separation of concerns between client and server, built-in security and authentication features and the ease of integration between different interfaces, in this case a web application and RESTful API. This section presents an exploration of concepts related to web services and technologies to help build them.

A web service can be described as a communication interface, typically using the HTTP protocol, and is meant mainly for machine-to-machine communication. Web applications often interface with an underlying web service in order to interact with its backend. The two most popular philosophies for building web services are SOAP and REST.

```
1
2 getUserREST(userId){
3     result = request.get("https://example.com/user/"+userId);
4 }
5
6 getUserSOAP(userId){
7     result = request.get("https://example.com/getUserInfo.php", userId)
8 }
```

Listing 2.1: Example illustrating the conceptual differences between a REST and a SOAP API call.

SOAP (Simple Object Access Protocol) is a protocol that is based on exposing parts of application logic as a service to its users, while REST (Representational State Transfer) focuses on giving access to named resources. While these protocols are mostly interchangeable, meaning that most tasks can be achieved by using one or the other, there are certain reasons why one would use REST over SOAP, and vice-versa.

SOAP, which is described by Suda et al. [Sud03], was created around the year 1998, which means its an established protocol with specific boundaries and guidelines. It communicates using Extended Markup Language (XML) messages in a fully stateless way and has support for many extension protocols which offer extended functionality, such as WS-Security, WS-Trust, and WS-SecureConversation, which provide security mechanisms on top of SOAP. Due to the more function-oriented nature of the SOAP protocol, calls to a SOAP web service cannot be cached, since information most likely needs to be provided by the client in order to achieve the desired functionality.

The REST architecture, created by Fielding [FT00], has mostly taken over today's web services. Its more data-driven nature allows for a separation of concerns between client and server, since clients do not know and do not need to know what goes on behind a call to an API, they simply issue the call and receive the resource in return. This also allows for some API calls to be cached on the client side, assuming the call is made to a static resource. REST also allows for a multitude of data formats, such as HTML, JSON, and XML. JSON is mostly preferred as it lowers the bandwidth necessary to transmit information, though HTML is also widely used. This architecture implies that the API is loosely coupled to the underlying server, making it simpler for changes to be made on either side.

To give an example of the conceptual and behavioral differences between SOAP and REST, the code snippet presented in Listing 2.1 illustrates the way one would go about retrieving information about a user:

While in the REST version of the API one simply needs to name the resource one wants to interact with, in the SOAP version one needs to specify the action of retrieving the information and give it the information necessary to identify and retrieve the wanted information. In REST, actions are specified by using the different HTTP messages available: a GET request implies information

retrieval, while a POST request implies the creation of new data and a PUT request implies an information update to a resource.

In conclusion, while most tasks can be achieved via either SOAP or REST, the norm today is to use REST unless there is a valid reason to use SOAP. The reason for this is that REST is simply more convenient and easier to use as an architecture than SOAP as a fixed protocol. Some of the main advantages of REST over SOAP include:

- REST uses significantly less bandwidth to transmit messages than SOAP;
- It can use JSON, which is one of the most used data formats nowadays;
- Separation of concerns between the API and the server makes it easier to make changes when necessary.

Some advantages of SOAP over REST are as follows:

- SOAP has built-in ACID compliance;
- It has security extensions, such as WS-Security, which make it more secure than REST;
- Unlike REST, SOAP is protocol independent, meaning that SOAP messages do not need to be sent over HTTP.

2.3.3 Web Applications

2.3.3.1 History

Web applications have seen considerable evolution since the inception of the World Wide Web (WWW), in 1989 and all throughout the twenty-first century. Four main phases of this evolution are identified, from web 1.0 to web 4.0.

Web 1.0 is the name given by various authors [ANF12, ZP] to the first generation of the web and web applications. This generation consisted almost uniquely of read-only websites. Websites could provide content such as catalogs or brochures for users to read and interact through emerging shopping-cart applications. Users were not supposed to interact with the web pages at all. Pages were static and suffered infrequent updates. From a technological standpoint, Web 1.0 applications consisted mostly of simple client-server applications where the client contained no business logic and was only responsible for rendering the page that was given to it by the server. Interactions between client and server were comprised of page and media requests by the client and subsequent responses by the server.

Web 2.0, as described by authors [ZP, Mur07, ANF12], brought about a change in the web. A paradigm shift occurred and new types of web applications, no longer read-only but bi-directional read-write applications, started to appear. Through these applications, users were able to interact and share content freely. Communities started to form on the web, with various purposes such as the gathering and sharing of knowledge. Users could be notified whenever a web page of

their interest was updated. Technologically, the web 2.0 era also marked the birth of several development tools for web applications. Using Asynchronous JavaScript And XMLHttpRequest (AJAX), web pages were able to retrieve small pieces of information from servers to update web sites as a user interacted with them, providing a much more pleasant experience. A number of templating technologies for the web also emerged, such as wikis and blog websites, with which a user was able to create a page without needing to know about the underlying technologies, thus forming a more people-oriented web.

Web 3.0, also known as the semantic web, is described by authors [ANF12, BLHL01] as a web that promotes the ability of data being machine-readable. This can be achieved by defining a structure for data and providing links between concepts, so that machines can process these relationships and establish links between different datasets.

Finally regarding web 4.0, which we are currently on the verge of, it is described by authors [ANF12] as the generation of the web that connects everything, from humans to machines, promoting the symbiosis between all entities. The concept of the Internet of Things is closely related with the web 4.0, as it consists in making everything controllable through the Internet and having machines react to stimuli by human or machine interaction.

2.3.3.2 Web application frameworks

Web application frameworks help developers define and implement a web application. These frameworks provide various functionalities, such as templating engines for code reuse, Object-Relational Mappers (ORM) for database interaction and specification of API endpoints with relative ease.

Most of today's web application frameworks implement the Model-View-Whatever pattern, as seen in [Jaz07, ZP]. This pattern evolved from patterns such the Model-View-Controller (MVC) pattern and its evolution, the Model-View-ViewModel (MVVM) pattern. These patterns offer three components: The model, which is responsible for representing the data stored in the database, the view, which consists mainly of web pages and a wildcard, which is responsible for preparing the data from models to be used in views. In this section, some web frameworks that follow these patterns are presented and analysed.

Ruby on Rails [Rub] is a web framework which follows the MVC pattern. To this, it adds additional functionality, such as *scaffolding*, *ActiveRecord*, *Migration*, and *routing*. *Scaffolding* is used to automatically generate models, views, and controllers based on a database's schema. *ActiveRecord* is an ORM which enables users to interact with the database by handling only Ruby objects, therefore ensuring operation safety and saving the developers of the burden of having to learn SQL or any other query language. *Migration* is a way for Ruby on Rails to manage changes to the database's schema automatically. With this, the database's topology can be changed by simply changing the models. Finally, *routing* maps a Universal Resource Locator (URL) query to a controller and action. This enables easy specification of API endpoints. [Jaz07].

Another web framework is Django [Dja]. Django is the Python equivalent of Ruby on Rails, in the sense that it can do everything that Ruby on Rails can. The community surrounding Django,

however, offers a much vaster and stabler array of add-ons, plugins and general support than that of Ruby on Rails. Python as a language also forces a more explicit and correct programming style, meaning that Django's code, despite being more verbose, is easier to understand and debug.

Then, there is ExpressJS [Exp]. This framework is a NodeJS package which supports many of the features that the two aforementioned frameworks support. The use of JavaScript for both frontend and backend eliminates the overhead of handling and debugging two different languages. Furthermore, ExpressJS is an event-driven framework, which performs better for weaker hardware than its thread-based counterparts. Being an npm package, ExpressJS developers have at their disposal all packages that reside in one of the most active and prolific development communities. ExpressJS is also part of the famous MEAN stack, introduced below, which means it has a large user base and therefore is well supported.

The frameworks presented up to this point are frameworks for full stack web development. The following framework are frontend frameworks, which provide functionality to help create client-side web applications.

Angular [Ang] is a popular open-source frontend framework developed by Google and is the successor to the also popular AngularJS. Angular is based on Typescript, a superset of ECMAScript6 (ES6) with backwards compatibility with ES5. This framework provides various functionalities, such as a feature full templating engine, which compile asynchronously, and dependency injection mechanisms, which help keep dependencies in order and helps in testing by mocking dependencies. Angular is used by many other tools to build applications, even for other platforms. Examples of that are Ionic and Ionic 2, which used AngularJS and Angular, respectively, to make cross platform applications for Android, iOS, and desktop.

ReactJS [Rea] is described by [ZP] as an open-source frontend library created and maintained by Facebook. It differs significantly from other frameworks such as Angular. By using ReactJS, developers need to write their applications as an agglomeration of reusable components, which can be used in other components to build bigger and more complex ones. These components are typically made by specifying the View's logic in the overridden *render* function of ReactJS's *Component* class with a JSX object. ReactJS provides functionality which minimizes the amount of Document Object Model (DOM) operations necessary whenever a change is made to a View. This means that it has better performance than other frontend frameworks. This does, however, come with some caveats. To identify the set of minimum operations necessary to get from one representation to another, the component's Virtual DOMs need to be compared and reconstructed from scratch. This operation is slower as the Virtual DOMs increase in complexity. Moreover, this implies keeping Virtual DOMs in memory, which may hinder the normal behaviour of the system.

Finally, KnockoutJS [Kno] is a lightweight frontend framework that is considerably smaller than other frameworks such as Angular, at the expense of offering only a subset of the features other frameworks offer. KnockoutJS provides developers with the essentials that make it an MVVM framework: declarative templates, data-bindings and automatic page updates given changes to the bound model.

Certain combinations of web-oriented technologies are so popular and so widely used that they

get their own nomenclature. The two most famous examples of this are the LAMP and MEAN stacks. The LAMP technology stack, which has Linux as the operating system, Apache as the web server, MySQL as a database and PHP as a backend. This stack is somewhat dated: At the time when it was popular, there were no choices for frontend frameworks other than pure javascript and jQuery. The MEAN stack is a more modern approach to web development, which has MongoDB as a database, ExpressJS as a backend system and web server, AngularJS as its frontend framework and NodeJS to provide any utility feature that might be necessary. The reasons why none of these stacks were used are manifold. Among them are the fact that MongoDB is not suited for this project and the fact that even though MEAN is a more modern stack when compared to LAMP, it is now also getting outdated, as there are many frontend and backend frameworks to choose from which offer the same if not better features.

2.4 Conclusions

This chapter described the state-of-the-art of some of the fields related to this dissertation.

Regarding network experimentation frameworks, in Section 2.1, one can conclude that despite their convenience and vast array of functionality offered in the field of network experimentation, their repeatability and reproducibility can be compromised due to external phenomena that can affect the output results of an experiment for the same given input. CONCRETE is used to try to achieve this repeatability and reproducibility, but even with this tool, only stable experiments can be deemed repeatable and reproducible. This dissertation proposes to go further by also being able to reproduce corner case experiments.

Concerning trace-based network simulations, Section 2.2 provides a description of the various approaches that exist in the state-of-the-art. Application layer replay systems focus on representing the traffic of the application layer, discarding the underlying layers. Packet-based replay systems concern themselves with the reproduction of a trace containing full packet flows. Physical layer replay systems control the physical layer according to the information provided in traces, while leaving the upper layers for the simulation models to control. Given the capabilities of the ns-3 network simulator, which does a good job of simulating the upper layers but leaves room for improvement in terms of accurately simulating the physical layer, the physical layer replay system seems to be the most suited to this dissertation's objectives.

After that, this chapter provided explanations of some of the concepts that could be necessary to implement this dissertation's proposed solution as well as compared some of the technologies that implement them.

A comparison of relational and NoSQL databases was made, detailing the differences and similarities, advantages and disadvantages between them. The conclusion was that NoSQL can provide better performance for use cases where there is a large amount of data involved, high rate of CRUD operations or the need for an unstructured or semi-structured dataset arises. Furthermore, a comparison of some of the most popular NoSQL databases was done, detailing their strengths

and their weaknesses. From this comparison, it was possible to exclude those databases which offer poor write performance, scalability or do not offer durability.

The two main philosophies for developing web services were also compared in this chapter. The conclusion was that the data-driven philosophy REST has been the choice of an overwhelming amount of developers for the implementation of web services, and that it is also better suited in the case of this dissertation's proposed solution.

Finally, the history of web applications was briefly exposed, after which the concept of a web application framework was explained and a comparison was made between a number of existing frameworks. From this comparison, the conclusion was that most frameworks offer features that significantly facilitate the development of a robust web application.

Chapter 3

Architecture and Design

This chapter expands on the difficulties of gathering all of the requirements necessary to correctly and easily perform trace-based network simulations. It then explains how the process of obtaining a trace-based simulation can be automated. The solution proposed by this dissertation is then presented from a high-level architectural perspective.

3.1 Requirements for trace-based network simulations

As explained in Chapter 1, trace-based network simulations differ from regular simulations in that instead of only using simulation models to approximate the network elements' behaviour and environmental conditions that would be encountered on a real testbed, selected data captured in previous real testbed experiments is used.

The data that needs to be captured differs between each trace-based simulation approach. For instance, application layer replay systems, introduced in Section 2.2, require data traces consisting of the kind of transport protocol used, the length of the packets and timestamps, in order to determine certain interaction delays. On the other hand, for physical layer replay systems, the information one must feed the simulation consists of the Received Signal Strength (RSS) sensed by each node for each other node, the noise floor, and the Global Positioning System (GPS) coordinates of the nodes. These coordinates are associated to a timestamp in order to enable the reproduction of the nodes' movements.

This dissertation will focus on creating a system that would automate the process of capturing the necessary data traces from real experiments and setting up physical layer trace-based network simulations. However, the system should be generic enough that extensions for it can be built in the future which could enable, among other things, support for the preparation of other kinds of trace-based simulations.

Capturing the necessary data traces required by a physical layer replay system is, as previously stated, a complex and error prone process. First of all, the network nodes must be running programs to capture the necessary data and store it internally. This can pose a problem if the duration of the experiment is such that the gathered data's size would exceed the storage capacity of the nodes.

Furthermore, the retrieval of the data must be done by accessing each node individually to collect its captured data. This process can be significantly time consuming depending on the number of nodes and can also lead to errors from data mislabelling, leading to inconsistencies in the data set.

Having all the information in one place, the developer must then manually edit and annotate the data set. This is necessary due to differences in some network devices' settings and units of measure. Once again, this process can lead to human errors.

Finally, the information must be incorporated into the simulation, which will require formatting the data to comply with the simulation's accepted format. Once again, the overhead and proneness to human error inherent to this process are considerable.

3.2 Proposed Solution

As stated earlier, the main goal of this dissertation is the creation of a framework that is able to automate the process necessary to reproduce real experiments in a simulated environment. This framework should also be robust to failure, perform efficiently in order not to disturb the real experiment, and secure, so that no data leaks happen and no experiment is compromised. In order to achieve this, the proposed solution is a multi-component system akin to what can be seen in Figure 3.1. Essentially, what is provided is a system which handles the automatic capture of traces to be used for trace-based simulations. This system also provides storage for the captured traces, a means of accessing the traces from each user's computer, and the automatic construction of pre-configured, plug-and-play trace-based simulations.

This approach should mitigate the overhead associated with the whole process. It also minimizes the number of unsupervised interactions the user needs to have with the system, which leads to a lower chance of human error.

This system also provides an easier way of validating experiment results, since it gives open access to the data and can generate pre-configured simulation files without requiring any knowledge of the underlying technologies from the user.

The following sections will elaborate on the proposed solution by analyzing the views associated with the 4+1 view model. The 4+1 view model is a tool to thoroughly describe a system through the use of multiple concurrent views. This view model is widely known and used to describe complex, multi-component systems such as the one comprising the proposed solution of this dissertation.

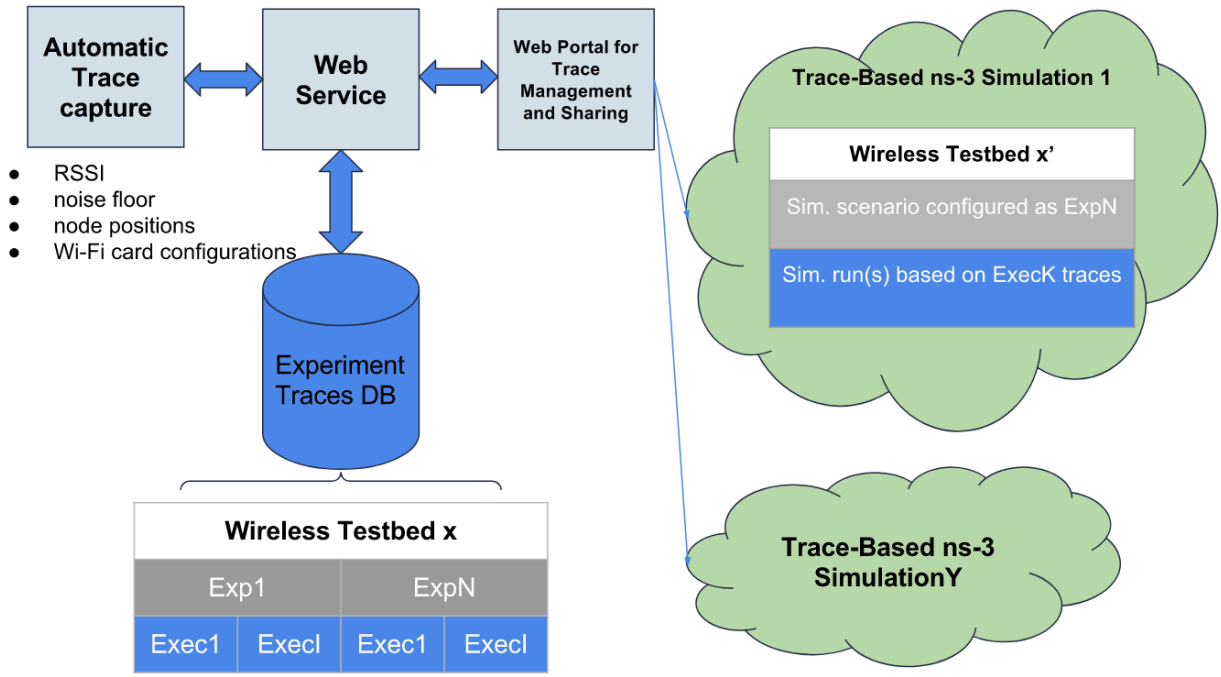

Figure 3.1: Sketch of the proposed solution's architecture.

3.2.1 Implementation View

The implementation view describes the system from a programmer's standpoint. It deconstructs the system into software components that can be implemented separately. The diagram used to provide an implementation view is typically the component diagram.

This dissertation's proposed solution's component diagram can be seen in Figure 3.2. The diagram introduces three main components: The information retrieval agent, the REST API and the Web Application. Each of these components is hosted on a different subsystem. The information retrieval agent is deployed within each node in an experiment. Its job is to retrieve relevant metrics as specified by the user. It is important to note that since the main objective of this exercise is the reproduction of the experiment in a simulated environment, some metrics are always extracted. These metrics are:

- The Received Signal Strength Indicator (RSSI) sensed by a node for each other node;
- The noise floor of the environment;
- GPS coordinates of each node, associated to a timestamp in order to be able to reproduce a node's movement.

The REST API is a part of the central storage subsystem and is mainly used to interact with the database. This component interacts with the information retrieval agents in order to receive data from these and also to provide synchronization and signalling commands. It also interacts with the user interface in order to give access to the data stored in the database.

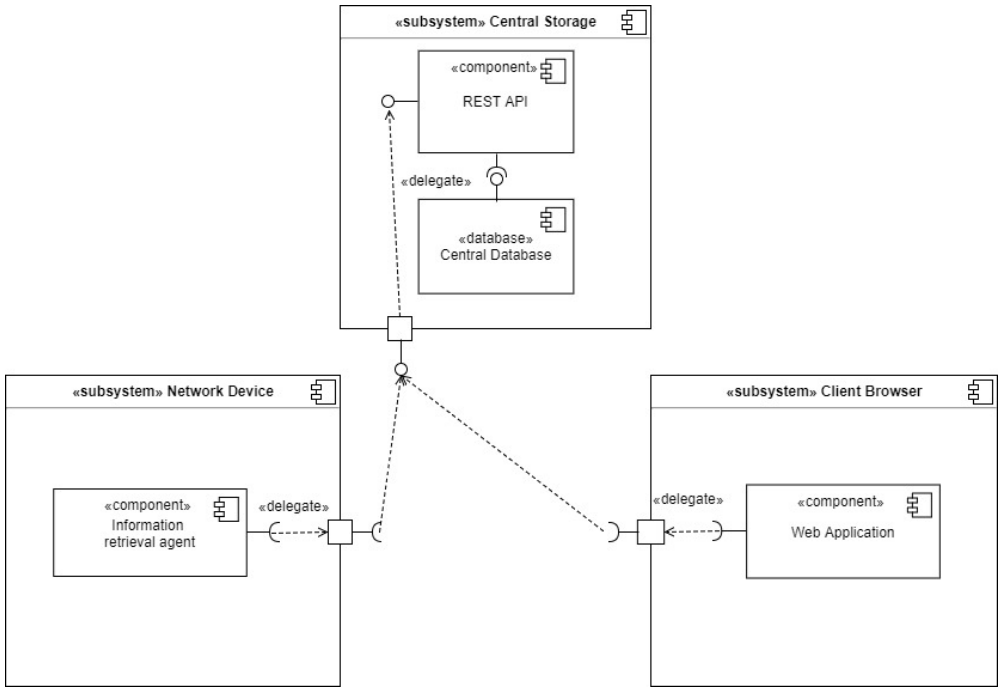

Figure 3.2: Component diagram describing this dissertation's proposed solution.

Finally, the web application is the component with which users interact. Being a web application, this component runs in the client's web browser. This makes the component easily accessible from any device with support for a web browser. This application offers data browsing, filtering and downloading capabilities, as well as the possibility to download a pre-configured ns-3 trace-based simulation based on the experiment the user chooses to reproduce. This component also offers user management and data processing and approval features to system administrators.

3.2.2 Physical View

The physical view describes the system from a system engineer's standpoint. It shows the topology of the different software components and their respective physical containers. The view is usually provided via a deployment diagram.

This dissertation's proposed solution's deployment diagram can be seen in Figure 3.3. As stated before, the information retrieval agents will be deployed in each network node. Both REST API and web application will be hosted on the same machine, since they can be considered back-end and frontend of the same web platform. The database will be hosted on a remote machine in order to promote the possibility of horizontal scalability without compromising the performance of the rest of the platform.

3.2.3 Logical View

The logical view describes the inner workings of the system's components. This is usually done using class diagrams. In this view, two class diagrams were made, one for the information retrieval

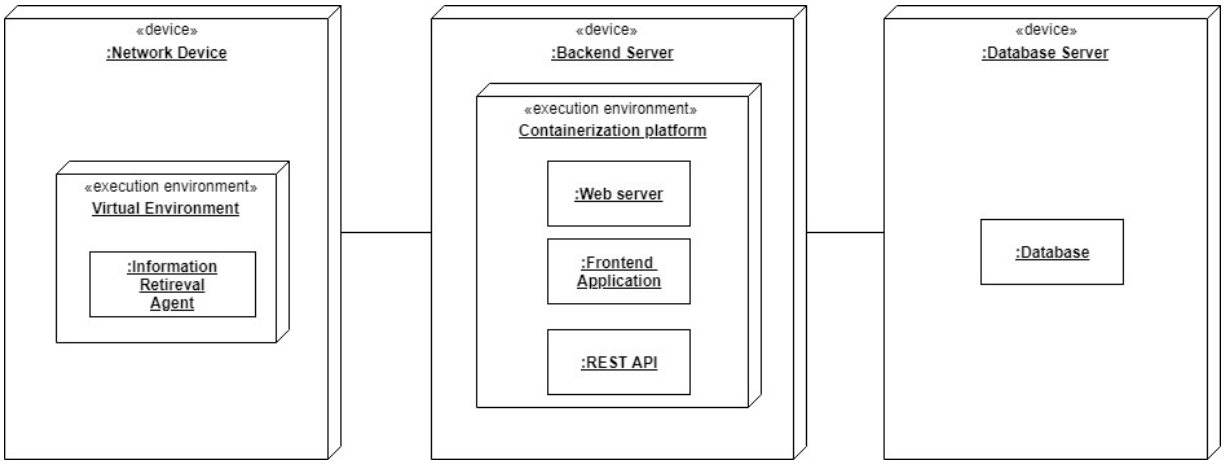

Figure 3.3: Deployment diagram describing this dissertation's proposed solution.

(Figure 3.4) and one for the frontend web application (Figure 3.5).

The information retrieval agent is composed by an arbitrary number of data capturing modules, one for each data stream that needs to be captured. It also contains a data manager, which is responsible for sending the data to the central storage system whenever the agent deems it appropriate. A number of utility modules are also part of the program, such as the setup module, which allows users to build the agent's configurations for the experiment and the database interface module, which abstracts the access to the agent's local database. There can also be a synchronizer module which synchronizes the node's clock according to a given source. This is helpful to ensure that data from multiple nodes is consistent, but is not necessary since clock synchronization can be done through the network implicitly. Finally, this component also contains a launcher module which initiates data capture and activates the agent's deliberation and data management capabilities.

The web application's diagram shows a typical Model-View-Controller architecture, where controllers manipulate the views based on information from models. In this case, the login and registration pages interact with an authentication service, which then interacts with the central database to manage authentication tokens and user permissions. Pages related to experimental data interact with a service whose sole purpose is to interact with the database so as to query the dataset according to the user's preferences. Finally, the administration page interacts with a service which is used to interact with the database in order to act upon users and user permissions.

No class diagram was made for the remaining component, the REST API, since this component is built using backend web frameworks which automate most of the boilerplate and structural decision making process.

3.2.4 Process View

The process view describes the system as the interaction between its components at runtime. This view can be described using an array of charts, one of which is the sequence diagram, showing the messages exchanged between components in the given situations.

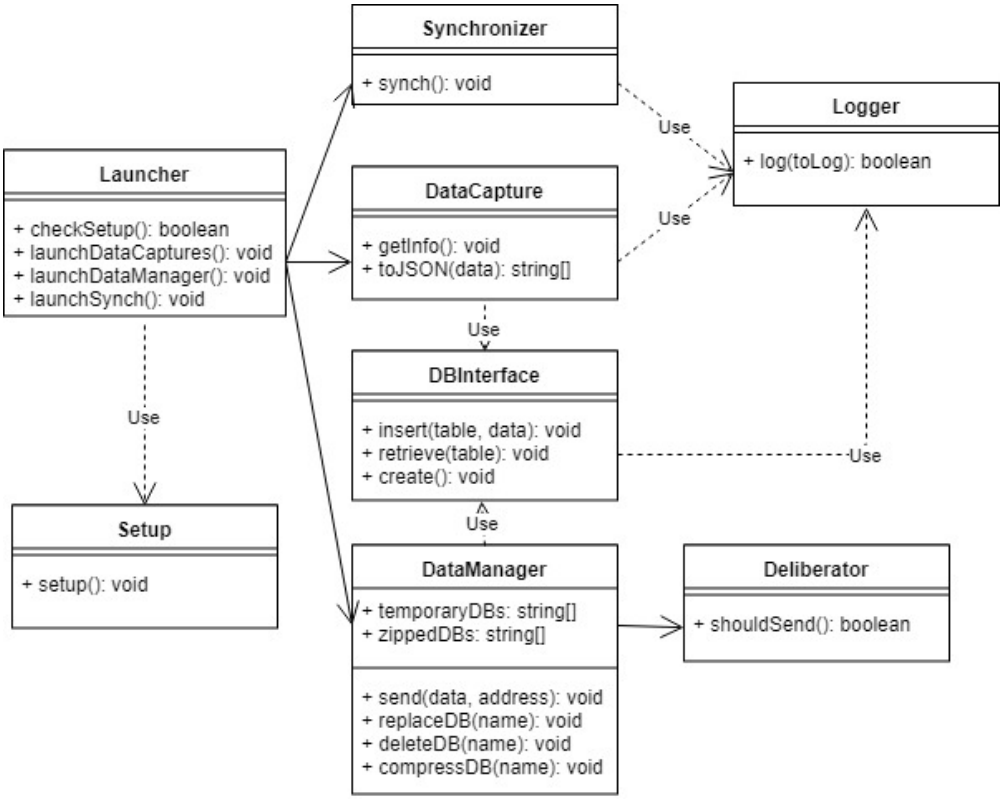

Figure 3.4: Class diagram describing the information retrieval agent.

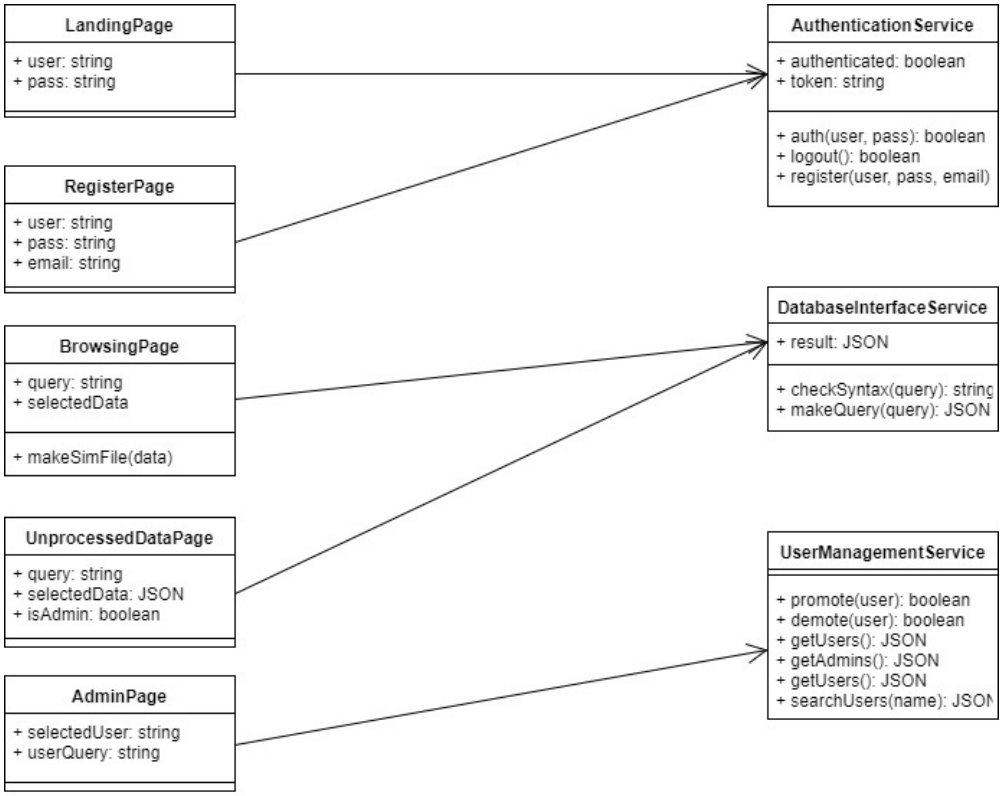

Figure 3.5: Class diagram describing the web application.

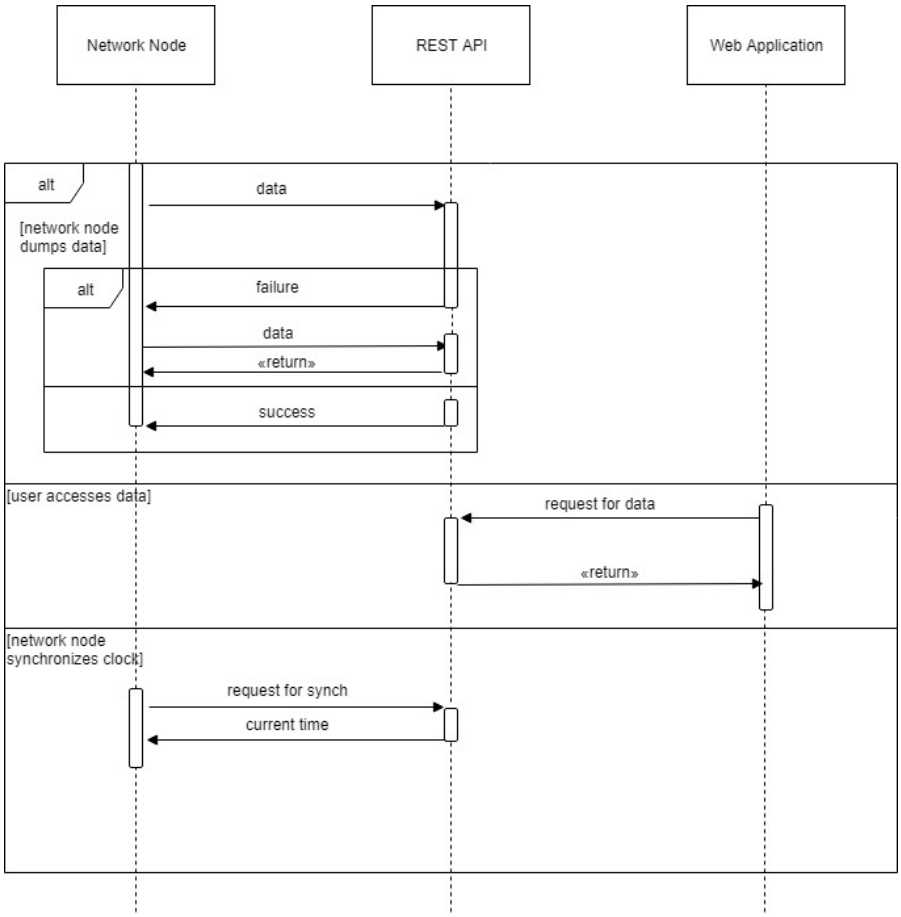

Figure 3.6: Sequence diagram describing this dissertation's proposed solution.

The sequence diagram for the proposed solution can be seen in Figure 3.6. The three main interactions between the system's components are depicted: the case in which the network nodes dump information to the central database for permanent storage, the case in which a user accesses data, and the case in which the network nodes synchronize their clocks with the host of the central storage system. The central storage system does not need to be the source of synchronization for the network node's clocks. However, since it is already a part of the system, it would be simple to extend the REST API to provide that functionality.

One can observe from this diagram that, upon failure to transmit the data between network node and central system, the node will continuously retry the transmission and will only delete its data upon successful transmission. Other interactions are fairly straightforward.

3.2.5 Use Case View

The final view of the 4+1 architectural view model is the Use Case view. This view relates all other views by showing exactly what functionalities are provided by the system and which user classes will benefit from which functionalities.

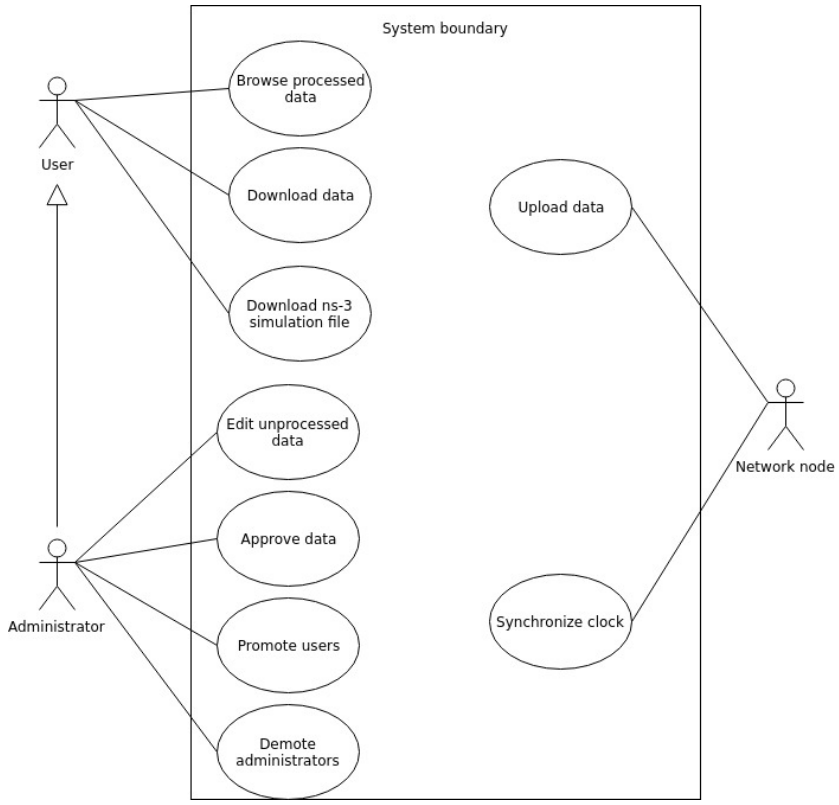

Figure 3.7: Use case diagram describing this dissertation's proposed solution.

The use case diagram for the proposed system can be seen in Figure 3.7. Three main user classes are defined: 1) the regular user, which interacts with the system through the web application; 2) the system administrator, which is a superclass of the latter class; and 3) the network nodes. The network nodes share the particular characteristic that they interact with the system directly through the REST API, instead of through the web application.

A regular user will interact with the platform in order to browse, filter and download data and pre-configured trace-based ns-3 simulations, while administrators have the ability to promote users to administrators and demote administrators back to users. Administrators also have access to the unprocessed data in the database in order to examine and edit it, if necessary, and approve it for regular users to see and download.

Network nodes interact with the platform in order to dump their temporary data stores and also to synchronize their clock with the REST API server's clock. This feature is important to guarantee that the node's movements are accurately reproduced. A slight deviation in a node's clock might result in radically different results. However, as stated before, it is possible to force clock synchronization implicitly through the network.

3.3 Conclusions

This chapter presented a more detailed description of the problem to be tackled in this dissertation project. It also presented a high-level description of the proposed solution through the use of the 4+1 Architectural View Models. The proposed solution is a framework that allows users to automate the process of gathering, processing, and reusing data to reproduce experiments in a simulated environment. The framework is comprised of three main components: an information retrieval agent, a REST API, and a web application. The information retrieval agent captures and transmits data to be stored in the central database. The REST API gives access to said database, for storing, accessing and user management. The web application allows users to browse and download data as well as pre-configured ns-3 trace-based simulations.

The objective of the system is to minimize the overhead associated with the processes described above automating most of the work. This solution also minimizes the number of unsupervised interactions the user needs to have with the network nodes and their data in order to reduce the risk of human error.

A secondary objective of this framework is to provide easy result validation. By giving open access to the data and providing pre-configured trace-based simulations without the user needing to have solid knowledge about the underlying concepts and technologies, this system provides the means for people to easily verify experimental results.

Regarding the non-functional requirements of robustness, lightweightedness, and security, these are not guaranteed at the architectural level, so that alternative implementations can be developed which cater to different needs. Nonetheless, an implementation of this architecture can easily fulfill those non-functional requirements, depending only on the technological and development choices and the quality of the implementation.

Chapter 4

Implementation

This chapter elaborates on low-level details that are relevant to the implementation of the proposed solution, such as the technology choices made for each component of this project, as well as a description of their inner workings and features. It is worthy of note that not all features described in Section 3.2 were implemented. The current state of this project consists of a proof-of-concept prototype with a set of key features that allows the system architecture to be validated, while fulfilling its main goals. This proof-of-concept prototype was built so that it can easily be improved in the future.

4.1 Information Retrieval Agent

4.1.1 Chosen Technology

Technologies that aid in the implementation of agents tend to provide an excess of features, such as graphical user interfaces and containerization facilities. Since this component of the system is meant to be run on network nodes, whose processing resources are limited, without impacting the system's performance negatively, the technology used to implement the agent needs to be fairly lightweight. For that reason, the Python programming language was chosen to implement this component, using almost no external libraries. Python provides a very complete toolset, which allows one to achieve complex tasks quickly. It is also a language which, albeit not as efficient as languages such as C/C++, offers enough performance capabilities to make its ease of development preferable over other language's performance.

4.1.2 Implementation

This retrieval agent was implemented considering future expansions. It is composed of a number of modules:

- A launcher module;

- A setup module;
- A database interface module;
- A data management module;
- A deliberation module;
- Three data capturing modules.

The agent also requires the presence of a SQLite database file as well as a configuration file. This configuration file is a JSON file and was modeled so that it could be easily understood, edited and even copied to other nodes. It contains information such as the experiment's identifier, the address to which data should be sent, and the deliberator module to be used, among some others.

The launcher module is the entry point of the program. It first checks for the existence of the configuration file. If that file is not found, a call to the setup module is made and the setup process is begun before the program can advance any further. Upon completing this process, the launcher will start multiple threads corresponding to the data capturing modules as well as the data management module. At this point, the information retrieval agent is initialized and working.

The setup module offers a simple text interface for building the aforementioned configuration file and also creates and initializes the SQLite database in the process. This module can be run either as a part of the agent's normal execution, as mentioned above, or separately, giving the user the option to generate the configuration and database files from its own computer. This is useful, as configuration files are likely not to differ significantly for nodes within the same experiment. As such, the user can copy the same configuration and database files to each node and perform minimal edits to have the node ready for information retrieval.

The database interface module acts as a buffer between the other modules and the database. Although the presence of this module potentially hinders the modularity of the information retrieval agent by creating a dependency between each data stream and itself, it acts as a black box for interaction with the database, allowing for boilerplate database access code to be reused instead of having it copied for each data stream's module.

The data management module is where the data in the SQLite database can be sent to the central storage system. It uses the deliberation module specified in the configuration file to ascertain when data should be sent. This means that users can make their own deliberation module to replace the agent's default one according to their needs. The default deliberation module will flag the data management module to send data after thirty seconds elapse since the last send, or when the database has accumulated 500 kB worth of data. When that happens, the data management module will atomically rename the database file, creating an empty one with the correct name, allowing the flow of data to continue with minimal interruption. The data is then extracted from the renamed database and formatted into a JSON array containing three objects, one for each data stream, which in turn contain JSON arrays with objects corresponding to the captured lines of data. This formatted data is then sent via HTTP to the specified machine. Upon a successful response,

this temporary database is deleted to save space, allowing for the information retrieval agent to gather data perpetually without losing local storage storage capacity.

The data management module also comes with a number of fail safe options in order to guarantee an uninterrupted flow of data even in the event of a network loss, or other issues, causing an inability to send the data to the specified machine. When this happens, a number of retries, specified by the user in the configuration file, will happen asynchronously. New temporary renamed databases receive a different name if the data management engine detects the presence of past temporary databases, ensuring that no data is misplaced. Finally, if none of the attempts at sending a batch of data succeeds, the temporary database will be compressed and left in the network node's storage for manual retrieval by the user.

Lastly, the data capturing modules are the components of the information retrieval agent that are less reusable. Each module captures a specific data stream by extracting the needed information from the system software that releases it. This means that when using this agent with different network devices, new modules may need to be developed or adapted from the existing ones, according to the available hardware and software. The three data capturing modules developed for this prototype were the following:

- GPS Data - The GPS data capturing module uses the *gps3* python library, which interfaces with *gspd*, a tool that parses and presents GPS data from the external GPS device attached to the network device.

- Wireless information - The wireless information data capturing module relies on the existence of the *iwinfo* program on the device. This program shows information about the device's network cards, such as noise floor and transmission power.

- Packet data - The packet data capturing module is dependent upon *horst*, a lightweight Wireless Local Area Network (WLAN) analyser. This module has a preparation phase in which it launches a thread which launches *horst* as a subprocess. It also creates a Unix pipe and directs *horst*'s output to that pipe, to be read and stored by this module.

In order to be able to atomically enter and retrieve data from the database, each data capturing module needs to receive two Python *Event* instances, one which it will control and the other one corresponding to the data management instance. The *Event* class offers functionality that allows for thread-safety in the information retrieval agent. Every time that one of the data capturing modules needs to write data to the database, it sets its own event and unsets it immediately after the write has been performed. Whenever the data management module initiates the process of sending data to the central storage system, it will set its own event and then wait for the events of all capturing modules to be unset. When that happens, it will perform the database swap and immediately unset its own event, allowing the data capturing modules to resume writing to the database. This behaviour ensures that no data gets lost during either the process of writing continuously to the database or during the process of swapping the databases.

Table 4.1: Description of each of the endpoints provided by the central storage system's REST API.

Endpoint	Allowed Methods	Expected input/output	Expected status codes
/data	GET, POST	[{'type': 'gps', 'data': [{...}, ...] }, {'type': 'info', 'data': [{...}, ...]}, {'type': 'packet', 'data': [{...}, ...] }]	200, 400
/gps	GET, POST	,[{ GPS data }, ...]	201, 400
/info	GET, POST	[{ info data }, ...]	201, 400
/packet	GET, POST	[{ packet data }, ...]	201, 400

4.2 Web Platform

4.2.1 Chosen technologies

Previous chapters explained that the solution proposed in this document is comprised of an information retrieval agent and a web application, which in turn consists of a client-side application and a REST API. In Section 2.3.3, a number of web frameworks are introduced and their characteristics explained. From the technologies that were mentioned in that section, the following were selected for the project.

The Django web application framework was used along with its Django REST framework add-on to implement the REST API that acts as the backend of the web application. The Django REST framework makes it relatively simple to specify and implement a working REST API. Furthermore, to expand on the description made in Section 2.3.3, Django itself is easily configurable, scalable, fast, and offers built in functionality for unit tests. Even though Django also offers a templating engine with which one would be able to build a frontend to the web application, this was decided against due to the fact that some of the more established MVC or MVVW web application frameworks are able to offer features that not only make it easy to add functionality to the application, but also significantly facilitate the deployment of this interface on a multitude of devices.

The Angular web application framework was used to build the web application that is the frontend of the web platform. The fact that this framework is able to produce code that works in cross-platform environments gives this project the ability to easily expand its interface into different platforms other than the proposed web application. Moreover, the way the Angular framework is built makes the implementation of dynamic web pages easy and, with it, unit testing can easily be achieved.

4.2.2 Implementation

The REST API built with Django and the Django REST API for the central storage system has four endpoints, as described in Table 4.1.

The exact metrics expected for each data stream are omitted from the table in order to maintain its legibility, but can be seen in Appendix A.

As stated in Section 4.2.1, Django and the Django REST Framework make it quite simple to define the endpoints of REST APIs. The Django Rest Framework provides template classes which allow one to bind an endpoint to a model, a representation of a table in the database. Using those template classes, the behaviour for the traditional CRUD operations is automated, meaning that there is no need to repeat the same code for each of the single data stream endpoints' GET and POST methods.

This is possible due to the definition of serializers. Serializers are also a feature provided by the Django REST Framework which automatically bridges the gap between data structures such as JSON and Django's database models. Therefore, in order to process requests that come into this component, one needs only to either pass the request body to a serializer in the case of a POST request, or to pass the desired model instances to the corresponding serializer to obtain a valid JSON representation in the case of a GET request.

For this project, the methods that handle the GET requests of individual data streams were overridden in order to provide filtering functionalities to the front-end application, such as filtering by date, node, or experiment. The remaining endpoint's methods had to be explicitly defined, since its behaviour is not part of conventional REST CRUD operations. This endpoint receives and sends data pertaining to all data streams at once. It is used by the information retrieval agent, so that it only needs to perform TCP and TLS connection setups once.

Django's ORM provides automated interactions between it and the most popular relational database systems: SQLite, PostgreSQL, MySQL, and OracleDB. In order to be able to interact with the Cassandra database used in this implementation of the system, the Django Cassandra Engine add-on was added to the project. This add-on provides functionality close to the one provided by the default Django ORM. However, certain actions are not supported, either due to add-on's relative youth or because Cassandra, being a NoSQL database, does not support such actions.

Finally, in order to have the REST API accept requests from external devices, such as the information retrieval agents, the Django Cross-Origin Resource Sharing (CORS) Middleware was also added to the project.

Regarding the web platform's frontend, the Angular project that comprises it is composed by a single page and three services which provide additional non-interface functionality. The page displays a table containing the data for the currently selected data type, as well as a small command module, as seen in Figure 4.1. The page's layout was built using Bootstrap, a Cascading Style Sheets (CSS) and Javascript framework for webpage styling.

One can switch the data type displayed in the table by using the tabs at the top of the page. Also, one can use the arrows or click the numbers to the left of the page in order to see more data. Pages are limited to fifty entries per page. It is also possible to change the way the data is displayed by clicking on the header of each of column in the table. Doing so makes the data reorder itself so that it is sorted according to that column, either increasingly or decreasingly. This feature is provided by the sorting service, which provides a method which, given a dataset and a comparator function, orders the dataset according to that comparator.

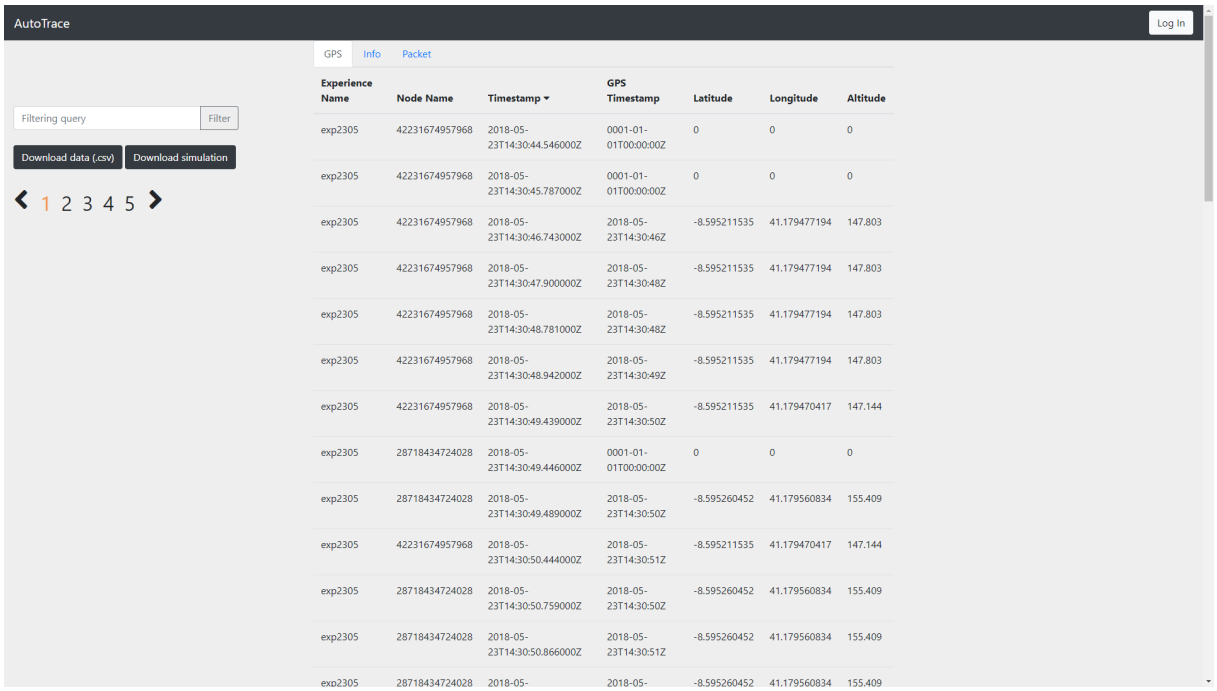

Figure 4.1: Look of the web application

The web application interacts with the backend using the aforementioned REST API. A GET request is made for each data type as the page loads. Also, every time the filter button is pressed, a GET request is made for the currently selected data type including the filters typed into the input by the user. These requests are made using the data service, which contains methods which issue said requests, building URL parameters dynamically according to the options specified by the user.

This application can dynamically generate a CSV file for the data type that is currently selected, including any filters that are applied. This can be done by clicking the "Download data" button.

Finally, this web application is able to dynamically generate the files necessary to perform a trace-based network simulation representing the specified experiment. By clicking the "Download simulation" button, a modal will appear where the user can specify the experience it wants to reproduce as well as input some settings, such as altering the transmission power, antenna gain or IEEE 802.11 standard to be used in the simulation. When all of those options are correctly filled in, a C++ ns-3 simulation file will be built according to the user's specifications, along with the necessary CSV files containing the node's GPS coordinates and SNR values. Further details on how to produce a trace-based ns-3 simulation are available in Section 4.6.

All file generation is done using the file service. This service offers a number of methods, the main ones being the csv preparation and simulation file preparation ones. The remaining methods are auxiliary functions which format datasets and infer csv headers from those datasets.

4.3 Proxy Node

This component, although not included in the specifications provided in Section 3.2, is necessary for projects where the network nodes might not have immediate access to the Internet. This node contains a REST API built with Django which offers the same endpoints as the central storage system as well as an additional endpoint which makes this node send all the data in its database to the specified address. This address can map to the actual central storage system, for storage, another proxy node, or any other device, for pre-processing or approval.

The idea behind this replacement node is to have a local temporary storage system, which can store the data sent by the nodes and send it in bulk to any other device at the user's convenience.

4.4 Database

4.4.1 Chosen technology

As explained in Section 2.3.1, the use of NoSQL databases is suited for this project, since most of them handle large amounts of data more efficiently and are capable of horizontal scalability. Furthermore, the ability to handle semi-structured data makes it so that users can specify the metrics they want retrieved from the network nodes.

The NoSQL database chosen for this project was Cassandra. This database provides high write performance as well as the ability to handle semi-structured data. Its shared-nothing architecture also allows for easy horizontal scalability.

4.4.2 Implementation

The database was modeled according to the three data streams that are recorded by the information retrieval agents and stored by the central storage system. Among that data are the metrics needed in order to produce a physical layer replay system, which are mentioned in Section 3.1.

4.5 Deployment

4.5.1 Chosen technologies

In order to make both components of the web platform available on the Internet, Nginx [ngi] was used. Nginx is a high performance HTTP server, which is widely used by a number of industry-leading companies, such as Vodafone and Visa. The tool offers production-ready web serving features, with added functionality such as load-balancing and content caching.

To make the whole web platform more easily deployable and modular, it was containerized using Docker [doc] containers. Docker is a container platform which is relatively simple to configure and offers a very complete crowd-sourced library of pre-built containers. By using this technology, the whole web application can be run from any computer by simply downloading the appropriate

docker configuration files and the project's source code and running the launch command, making the whole solution simple to deploy.

4.5.2 Implementation

Two modes of deployment were considered for this project, given the different purposes one may have in deploying the system. Separate Nginx and Docker configurations were made for development and production environments.

Development configurations are used when developing the system, to deploy the web platform on the user's local machine. Production configurations are used when deploying the web platform on a server open to the world. The production configurations have the frontend application compiled with optimizations and served by Nginx as static HTML and Javascript files, while the development configurations launch this application as a separate web-server to which Nginx transmits requests via reverse proxy. This is useful while developing the solution since the Angular web-server will recompile and refresh the page every time it detects a change in the project's files.

One other docker configuration exists to run the developed test batteries and relay the results.

4.6 Trace-based simulations

As described in Section 3.2, the solution proposed in this document should be able to produce pre-configured trace-based simulations.

ns-3 [ns-] is an open-source discrete event network simulator which offers simulation and emulation of all network layers as well as a large number of simulation models to account for the diversity of network technologies and scientific models that exist in the state-of-the-art. Being open-source, this simulator sees regular updates as an increasing number of network developers and researchers submit their own models to further develop the simulator. This simulator was chosen, not only because of its completeness, large feature set and supported simulation models, but also because it is one of the most widely used network simulators.

One contribution made to the ns-3 simulator was the addition of support for physical layer trace-based simulation. This support comes in the form of a new propagation loss model called *TraceBasedPropagationLossModel*. The main feature of this model is that it can take the received signal strength between two nodes and apply that value to produce the Signal-Noise Ratio (SNR) that best reproduces the experiment.

The mobility of the nodes is supported by ns-3 by default, using mobility models such as the waypoint mobility model, in which one defines all the points the nodes are supposed to go through and at which time.

Combining these two features and the information taken from the real experiment, one can accurately reproduce the experiment's physical characteristics by actively changing the variable containing the perceived RSS between the nodes to conform to the values taken from the past experiment.

4.7 Conclusions

In this chapter, the technologies chosen for the implementation of the proposed solutions were described. Cassandra was used as the database for permanent storage. Angular was used for the frontend web application and Django to implement the REST API. Both of these components are served via Nginx. The whole platform runs in a docker environment, in order to be easily deployable. This set of tools is one of many possible combinations of technologies that can be used to achieve a working implementationfe of the system proposed in Section 3.2. These technologies were chosen, however, through a process of analysis of which technologies would better suit the system's functional and non-functional requirements.

An explanation of each component's features and implementation details was also provided. The information retrieval agent is comprised of a series of detachable modules which allows users to easily reuse parts of its code when adapting it to work on different hardware. The central system consists of a REST API with 4 endpoints: one for each individual data stream and one to receive and output the three data streams simultaneously. The web application in the frontend displays the data available in the database and allows the user to filter it and save it to its machine as CSV files. It also offers the possibility of generating a fully automatic, plug and play trace-based ns-3 simulation representing the reproduction of the chosen experiment.

Finally, this chapter also presented details on ns-3 trace-based simulations. The main contribution that makes physical layer trace-based simulations possible in ns-3 is the addition of the *TraceBasedPropagationLossModel*. This model, in combination with the model that simulates the nodes' motions between different points and the information retrieved from the experiment, one can accurately reproduce the physical characteristics of the experiment.

Chapter 5

Validation

This chapter focuses on the validation of this dissertation's proposed solution. The framework was tested for its proper operation, mainly focusing on the offered functionality and performance. The impact of using the framework is also assessed. Each validation method and respective results are presented and discussed in detail.

5.1 Validation methods

The solution proposed in this dissertation was validated from three different standpoints:

- Functionality - Whether the system solves the intended problem by providing the necessary functionality and behaving according to expectations;
- Performance - Whether the system performs acceptably under large loads of requests and data;
- Impact and usability - Whether the solution constitutes a valuable asset in the field of network research.

5.2 Functionality

In order to assess the solution's adequacy to perform its intended purpose, unit and integration tests were run on its components, and a full system test was done.

For the information retrieval agent, unit tests were developed which cover low-level details, such as the correctness of both the database interface module's interactions with the database and the data management module's database file operations, among others. This test battery covers approximately 70% of the information retrieval agent's codebase. More coverage could not be achieved due to the fact that the data capturing models cannot be run separately from the rest of the system.

Regarding the REST API, Django's unit testing tools were leveraged to develop unit tests which cover all of its endpoints' execution. These tests cover the totality of the code related to endpoint behaviour and database interaction.

Finally, in order to validate the functionality and behaviour of the system as a whole, an experiment was carried out using a real testbed in which the two nodes involved had this solution's information retrieval agents recording and sending data to the central storage system. This experiment involved two Alix 3D3 network nodes, whose specifications are presented in Annex B, which were transferring data at their maximum capacity by using the *iperf* traffic generation tool to send User Datagram Protocol (UDP) traffic from one node to the other, one direction at a time.

Throughout the experiment, the system behaved as expected. All of the relevant data was sent successfully to the central storage system and was instantly available through the web interface, both for viewing and downloading. Despite the experiment's success, however, it also brought to light a potential issue with the solution: Since the nodes were transferring data at their maximum capacity, the amount of data generated by *horst* was of such magnitude that the thread which is responsible for running the packet data capturing module required more processing resources than expected. Due to the fact that the agent was running in a single-core environment, with very limited perfromance, the node's CPU started starving other threads in order to favour the aforementioned data capturing thread. This resulted in the wireless information data capturing module not being able to record data at precise intervals since it had to wait for the CPU to attribute processing time to it. A solution for this is to decrease the granularity of the data recorded in resource constrained network nodes. Instead of recording every packet generated, a user can record every n-th packet.

5.3 Performance

To evaluate whether the solution can handle large amounts of data correctly and in a timely fashion, load tests were performed, both on the central storage database and the REST API. Performance tests were not done on the information retrieval agent, as its performance is largely dependent on the hardware it runs on. Since this component is meant to run on a large number of devices, its performance on the current device is irrelevant.

The hardware used for these tests should be considered. The full specifications are available in Annex B. This hardware configuration is capable of supporting two Cassandra nodes at best, since an optimally functioning Cassandra node should use anywhere between 2 GB and 50% of its host device's RAM. Considering also the memory consumed by Django and the Operating System itself, there is only enough space for one Cassandra node to function properly. For this reason, the performance tests discussed in this section were performed in a single-node Cassandra instance.

Cassandra offers linear horizontal scaling. This means that adding a node to a Cassandra cluster always produces a concrete and calculatable improvement in performance. This is due to Cassandra's load balancing policies, which can have a designated coordinator node distributing requests among the cluster, therefore decreasing the average load of all nodes. Depending on

Table 5.1: Results of the write performance test on the central storage database.

Test case	Time to write
10	0:00:00.049220
100	0:00:00.353417
1000	0:00:03.532033
10000	0:00:36.256275

the load balancing policy, a query's consistency option, and the cluster's replication factor, however, read requests may display worse performance in multi-node environments, since Cassandra's eventual consistency makes it so that not all data is immediately available on every node in the cluster. Thus, a node may have to query other nodes in the cluster for the most up-to-date version of a table before answering the request instead of simply fetching the requested values, as would happen in a single-node cluster.

Both reading and writing performance tests were done on the database. These tests were performed twenty times and the results shown and discussed throughout this section consist of the average value for each test case.

For the write performance tests, an increasing amount of entries of all three data types were written to the database. Test cases consisted of 10, 100, 1000, and 10000 entries of each data type, resulting in payloads of 9 KByte, 88.9 KByte, 888.1 KByte, and 8.9 MByte, respectively. The results of these tests are presented in Table 5.1. These results show an expected linear increase in processing time as test cases get bigger.

It is worthy of note that in regular operation, situations will seldomly arise where an experiment has nodes uploading data concurrently at the scale of these tests' larger test cases. As explained in Section 4.1.2, the default algorithm for data management will have the information retrieval agent send data every 30 seconds, or whenever the database reaches a size of 500 KByte. The user can also specify its own custom algorithm to override this deliberation. It then falls in the hands of the user to ensure that an experiment does not cause database congestion. To avoid potential Denial of Service (DoS) attacks from the misconfigured deliberation modules, a central database system could, for instance, limit the number of write operations of an agent for a given period of time.

Furthermore, as explained above, performance would be improved in a fully fledged production environment, in which multiple Cassandra nodes would comprise the central storage system instead of just one.

Regarding reading performance tests, the test cases mentioned above with 10, 100, 1000 and 10000 entries of each data stream were used. The data was inserted into the database and then the time needed to retrieve all of the data was measured. Results can be seen in Table 5.2.

These results reveal an expected increase in processing time, proportional to the size of the data. This increase, however, is softer than the one shown by the writing tests' results. This is due to the fact that Cassandra has the ability to cache the data in a particular table as requests for it become more frequent. Unlike writing operations, however, the user has limited control over

Table 5.2: Results of the read performance test on the central storage database.

Test case	Time to read
10	0:00:00.030231
100	0:00:00.046427
1000	0:00:00.376507
10000	0:00:03.631749

the number of reading operations performed on the database. Therefore, one can conclude that, as a dataset gets bigger, the processing time necessary to retrieve it will eventually become unacceptable. This problem can be solved by two different approaches: 1) Implementing lazy-loading support on the API's endpoints; and 2) Adding more Cassandra nodes to the current cluster.

Concerning the API request performance tests, it was found that not only are the response times highly dependent on the amount of data sent or received, but they are also highly dependent on the state of the network through which the packets will travel. Since the state of the network is out of the user's control, these tests' results were deemed irrelevant.

Another metric that can be used to evaluate the solution's performance is its impact on the network. To that end, it is possible to analyze how much of the network's capacity is used to exchange data between the system's components.

For simplicity, we assume that only packet data is transferred, since it makes up around 95% of any given experiment's dataset. *Horst* outputs new information per each packet detected. For each Mbit/s (125 Kbyte/s) of traffic generated in the real experiment, around 90 packets/s are being captured by *horst*, considering the UDP maximum packet length of approximately 1400 Bytes. The amount of space occupied by the metrics deemed essential to be able to reproduce the experiment is 74 Bytes, which means that $74 * 90 = 6660$ Bytes of data are generated per second. This equates to $\frac{6660}{125000} * 100 = 5.328\%$ of the link's current load, which is an acceptable overhead. It is possible to generalize these calculations into a formula, as shown below:

$$overheadPercentage = \frac{\frac{1.25*10^5*linkLoad}{maxPacketLength} * lodSize}{1.25 * 10^5 * linkLoad} * 100$$

Where *linkLoad* corresponds to the given network's current load, in megabit per second; *maxPacketLength* is the maximum number of bytes a packet can have in the given environment; and *lodSize* is the number of bytes a line of data occupies. This formula can be further simplified so that its final form is as follows:

$$overheadPercentage = \frac{lodSize}{maxPacketLength} * 100$$

This formula reveals that, for a given maximum packet length and a given size for a line of data, the relative amount of network load introduced by the transfers required in the context of this system is a constant.

5.4 Impact

To perceive the way this solution will influence its target userbase's working method, a questionnaire was distributed to the network researchers at the Wireless Networks research group from INESC TEC, asking them to compare their regular workflow to the workflow they would have when integrating this framework into it. This questionnaire was made with Google Forms and distributed via e-mail. Ten answers were received, which corresponds to 100% of our universe.

The questionnaire starts by explaining what the objectives of this framework are and how trace-based simulations are used to enable repeatability and reproducibility of past experiments.It then goes on to describe the necessary steps to produce a trace-based simulation with and without the use of this framework. This preliminary discussion can be found in Annex C.

Followingly, a set of seven questions is posed to the questionee. The first two questions approach the time one thinks that it would take to produce a trace-based simulation with and without the framework. The results of the first and second questions can be seen in Image 5.1 and Image 5.2 respectively. Observing the pie chart representing the answers to the first question, one can see that the majority of people consider that it would take them more than a full day to produce a repeatable trace-based simulation which accurately reproduces the past experiment without the framework that is the product of this dissertation. On the other hand, the pie chart that represents the answers to the second question shows that half of the participants think that it would take them several minutes to achieve the same results, with the remaining half answering that it would take them either one or two hours (two responses), several hours (two responses) or one or two minutes (one response). From these two questions, one can conclude that this framework is considered by a significant part of the community that would use it to be a significant and impactful improvement upon the current methodology, namely by reducing the time required to complete the process from more than a day to, at worst, several hours.

The third and fifth questions tackle the possibility of human error involved in the processes, without and with the framework. The questionees are asked to rate that possibility in a scale from one to five, five being the most likely. The results to these questions can be seen in Figures 5.3 and 5.4 respectively. The bar chart that represents the answers to the third question shows that the majority of people answered between four and five, while a minority (two responses) answered three. Conversely, the bar chart representing the answers to the fifth question reveals that the vast majority of questionees answered with a two, while one person answered with a one and another answered with a three. Comparing these sets of answers one can conclude that the questionees believe that this framework also improves upon the original methodology in terms of eliminating the possibility of human error, namely from a four or five to, at worst, a three.

Questions four and six asked the people involved to identify the steps which they considered to be more error prone in the processes without and with the simulation, respectively. The answers to these questions are available in Annex C. The answers to the fourth question show a clear inclination towards both the step concerned with programming a specialized script to capture data and the step concerned with organizing and editing the data. Two questionees also mentioned

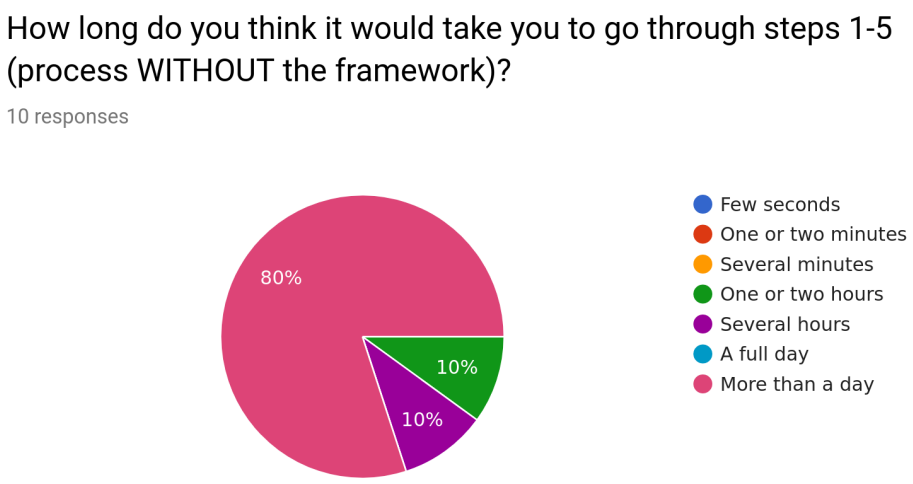

Figure 5.1: Pie chart representing the answers to the questionnaire's first question.

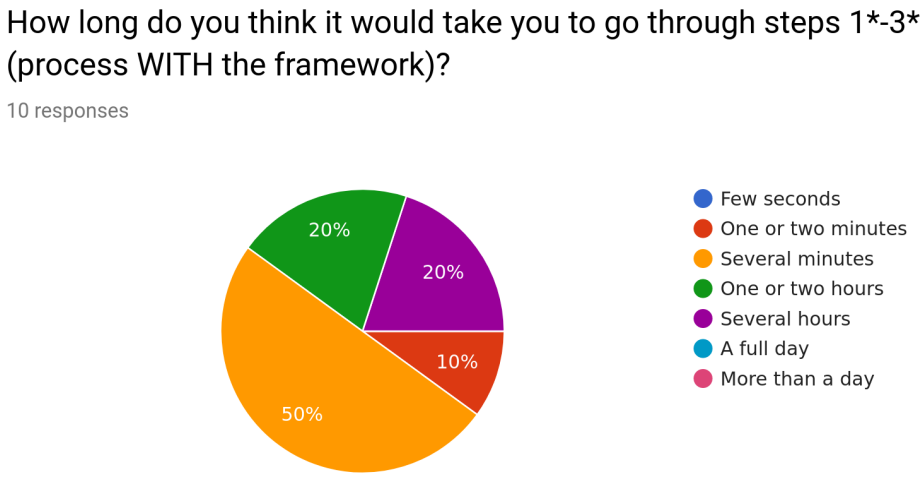

Figure 5.2: Pie chart representing the answers to the questionnaire's second question.

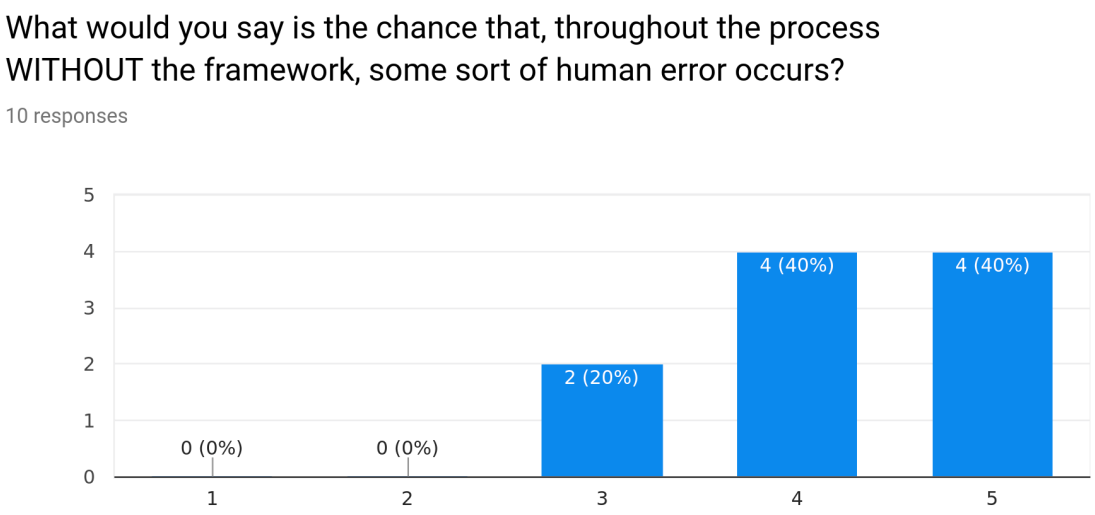

Figure 5.3: Bar chart representing the answers to the questionnaire's third question.

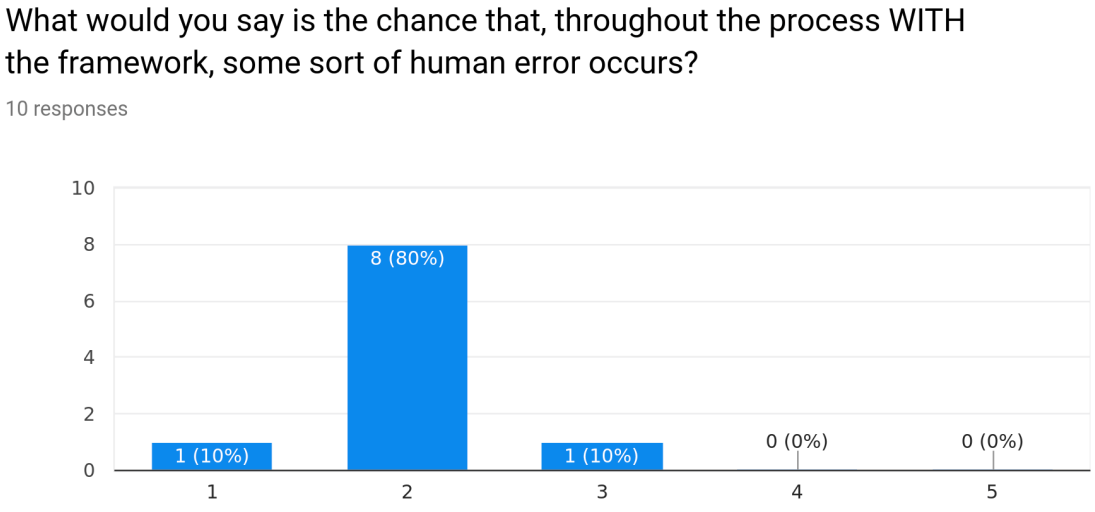

Figure 5.4: Bar chart representing the answers to the questionnaire's fifth question.

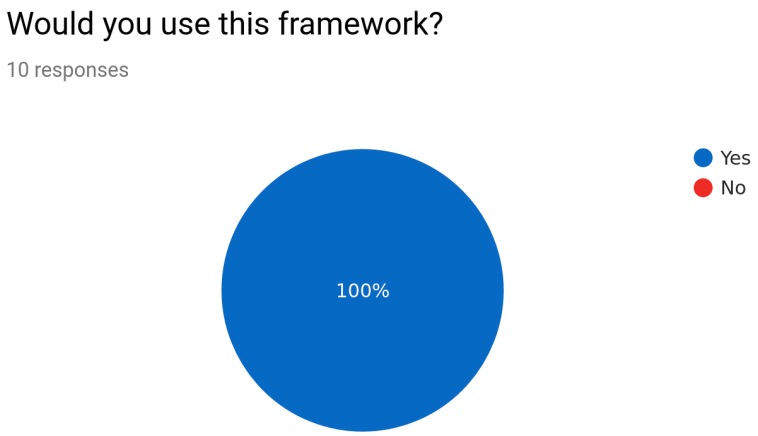

Figure 5.5: Pie chart representing the answers to the questionnaire's seventh question.

the step concerned with retrieving the data, while another one mentioned the step concerned with manually creating the ns-3 scenario. The answers to the sixth question present a clear tendency toward the step concerned with configuring the information retrieval agent almost unanimously as the most error prone step in the process. Only one user mentioned another step, namely the step concerned with downloading the pre-configured ns-3 simulation based on the selected experiment data. Analysing these two sets of answers, one can conclude that, despite having reduced the number of necessary human interactions, this framework does not completely eliminate the possibility of human error.

The final question in this questionnaire asked the questionees whether they would use this framework if they needed to use the trace-based simulation approach. The results can be seen in Figure 5.5. Every person involved answered that they would.

In conclusion, the results of this questionnaire show that the system proposed in this dissertation is considered to bring a significant improvement to the day-to-day operation and work methodology of the people from the INESC TEC Wireless Networks group The universe of potential interested people is expected to grow significantly once the trace-based simulation approach gets better dissemination in the ns-3 research community and also to the testbed providers and its users.

5.5 Conclusions

This section presented the methods through which this dissertation's project was evaluated as well as a discussion on the results of these evaluations.

First, validation of the system's functionality was approached. To verify that all components behave as expected, unit tests were developed and a real experiment was carried out. The components of the framework were developed so that the unit tests would pass. The experiment, serving as a final validation of the project's behaviour, ran within the expected parameters.

Followingly, the system's performance was evaluated. It was explained that performance tests on the information retrieval agent were not relevant, since its performance is largely dependent on the hardware it will run on. Nonetheless, since the tendency regarding the hardware capabilities of computer devices is to improve with time, it is expected that this component will not have any problems running in other nodes.

Regarding the database's performance, the results of the write tests revealed a linear increase in processing time accompanying the increase of data. This increase is acceptable since the user has fine grained control over when and how much data the information retrieval agents can send. As for the read performance test, it revealed a softer increase in processing time accompanying the increase in the amount of data. However, since the user has less control over read interactions in the database, this increase in processing time can become a problem as the available dataset increases. This issue is easily fixed by adding more database nodes to the cluster, or by implementing lazy-loading.

Finally, to verify that this project will have a positive impact on the network research community, a questionnaire was written and distributed among network researchers. The results showed that the framework is seen as an impactful improvement upon the methodology that is currently state-of-the-art, both in terms of saving time and in terms of reducing the possibility of human error.

Having the proposed solution successfully validated considering its functionality, performance, and expected impact on the research community unequivocally confirms the good quality of the solution and presents it as an important asset to the research community.

Chapter 6

Conclusions and Future Work

The main objective of this dissertation was to create a framework to automate the processes of gathering, processing and storing data from real network experiments and reusing that data in trace-based simulations to reproduce those same experiments.

This document starts by showing the problems that network researchers and developers encounter while testing a network solution through simulation and experimentation in real testbeds. Oftentimes, especially in emerging testbed scenarios, the results between simulation and experimentation, and even from multiple runs of the same experiment may differ greatly, rendering the experiments unrepeatable and unreproducible. Trace-based simulation provides a solution for that problem, allowing to reproduce past experiments in a simulation environment. Furthermore, there is no specific structure to the process of preparing this reproduction and the amount of interaction the developers need to have with the data increases the risk of human error. To solve this problem, a framework that handles these steps with the minimum amount of human interaction possible was proposed in this dissertation. This system also aimed to be robust, lightweight, and secure enough to operate as expected even in resource constrained testbeds, while keeping the gathered traces private during their transfer over the Internet.

Considering the requirements to implement the proposed framework, a preliminary technology review and comparison was made regarding the topics of database technologies, web services and web applications. The main conclusions were as follows: NoSQL can provide better performance for use cases where there is a large amount of data involved, high rates of CRUD operations or the need for an unstructured or semi-structured data set arises, such as the one encountered in this project. The database used as central storage in this system must have good write performance, scalability, and also be robust and reliable. A REST API makes more sense within the data-driven context of this system's interactions. Lastly, a vast amount of both backend and frontend frameworks exist which provide ample functionality and extensions. These frameworks will facilitate the development of a robust and easily deployable web platform.

The solution proposed in this dissertation was then presented from a high-level, architectural

point of view. The system is comprised of three main components: 1) the data retrieval agents, to be deployed on each node that is part of an experiment; 2) a web application consisting of a REST API, which interfaces with the database; and 3) a client-side application to act as a user interface. The 4+1 view model was also used to make an in-depth description of the system.

The solution's implementation details were then presented, by component, alongside the chosen technologies. The data retrieval agent, developed in Python, was built with modularity, expandability, and adaptability in mind. Its components are easily detachable and interchangeable. The REST API, built with Django and the Django REST framework, was developed to be robust and simple to use. The many features offered by Django and the Django REST framework make it easy to developed a fully fledged REST API which is tolerant to failure and can easily recover from any errors encountered during its execution. Finally, the web interface is able to dynamically generate a pre-configured, plug-and-play ns-3 simulation as well as the files containing the data traces for the desired experiment. It also shows the data that is available on the central database and allows a user to filter and download it. This component was built with Angular to ensure that its code is reusable and that the component itself can be made available not only in all PC operating systems, but also in mobile platforms.

Next, the proposed solution is validated considering its functionality, performance, and impact on the networking research community. In terms of functionality, the system provides the features necessary to fulfill the main goals of this dissertation. Performance-wise, the solution is able to scale if given the resources to expand the number of database nodes in a cluster, if needed. The interactions between this solution's components comprise a very small part of the network's load. Concerning impact, this framework is seen by the researchers at the Wireless Networks research group from INESC TEC to be a valuable asset, which significantly improves the agility and correctness of their work, while also fomenting the adoption of the trace-based simulation approach.

In conclusion, the successful validation of the proposed solution confirms that it fulfills all of the following requirements: Mitigation of user workload was achieved by automating almost all of the steps involved. Minimization of the possibility of human error was achieved by minimizing the amount of necessary user interactions in the framework. Regarding robustness, all components in the system are capable of failing gracefully and, in most cases, of recovering without hindering the system or somehow tainting the experimental data. In terms of efficiency, this solution's information retrieval agent is able to run on significantly constrained hardware without using too great a percentage of the device's resources. Furthermore, the interactions between this framework's components make up a small amount of the network load.

As a final note, although the security requirement was considered in the proposed solution design, due to the limited implementation time and its non-critical influence to validate the main functionalities, it was not considered in the solution's prototype. Nonetheless, some implementation notes are presented in Annex D on how to improve the system to fulfill this requirement.

6.1 Main Contributions

The main contributions of this dissertation are the following:

1. The development of a framework that automates the processes of gathering, processing and storing data as well as the generation of simulation code that allows for the accurate reproduction of an experiment in a simulated environment;

2. This framework introduced a significant impact on the research development (R&D) workflow as it reduces the amount of time and effort needed to adopt the trace-based approach, while also reducing the chance of human error. All of these advantages are culminating in the significant adoption of the tool;

3. The development of a platform that bestows upon its users the ability to share experiment data, in order to provide future users with more test cases for their projects, or to validate network projects results by granting access to test data without the need for prior knowledge on the topic of network simulation.

6.2 Future Work

Despite having achieved this dissertation's main objectives by producing a working solution for the problem tackled by this dissertation, there is a multitude of features that could be developed to improve its quality. Those suggested improvements are presented in Annex D, separated by component. These suggestions are significantly detailed and serve as potential guidelines for future implementations, targeting the evolution and expansion of the framework.

The tools provided by this dissertation's project will continue to be used and improved in the context of the SIMBED project. SIMBED is an ongoing H2020 European Project proposed by INESC TEC, which intends to make use of the testbeds made available by Fed4Fire+ in order to validate and disseminate the Offline Experimentation approach. This approach uses trace-based ns-3 simulations with the goal of achieving repeatability and reproducibility of experiments, exactly in the same way that the approach that this dissertation's proposed solution is exploring. The framework proposed and validated in this project will thus be used to facilitate the process of capturing, storing and reusing the data generated in the Fed4Fire+ testbeds mentioned in Section 2.1.

Also, independently from SIMBED, this framework is supposed to be shared among the scientific community. This would allow researchers to eventually be able to record their own data and access other people's data on the platform, either for data sharing purposes or for validation of results shown in scientific papers. This also introduces the potential of bridging together the ns-3 research community with other more experimentation oriented communities, which could leverage new synergies.

Finally, there are plans to integrate this framework and use it in the workflow of ongoing and future research projects in which the Wireless Networks research team at INESC TEC is involved. In fact, this is a testament to the potential it brings to real world applications.

References

[Aer] Aerospike. https://www.aerospike.com/. Accessed: 2018-02-08.

[ANF12] Sareh Aghaei, Mohammad Ali Nematbakhsh, and Hadi Khosravi Farsani. Evolution of the world wide web: From web 1.0 to web 4.0. *International Journal of Web & Semantic Technology*, 3(1):1, 2012.

[Ang] Angular. https://angular.io/. Accessed: 2018-02-08.

[AV16] Prakash Agrawal and Mythili Vutukuru. Trace based application layer modeling in ns-3. In *Communication (NCC), 2016 Twenty Second National Conference on*, pages 1–6. IEEE, 2016.

[avr] Apache Avro. http://avro.apache.org/. Accessed: 2018-06-22.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[Bre00] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[Cas] Cassandra. http://cassandra.apache.org/. Accessed: 2018-02-08.

[Cat11] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[Cou] Couchbase. https://www.couchbase.com/. Accessed: 2018-02-08.

[Dja] Django. https://www.djangoproject.com/. Accessed: 2018-02-08.

[doc] Docker. https://www.docker.com/. Accessed: 2018-02-08.

[Exp] ExpressJS. https://expressjs.com/. Accessed: 2018-02-08.

[FCR17] Helder Fontes, Rui Campos, and Manuel Ricardo. A trace-based ns-3 simulation approach for perpetuating real-world experiments. In *Proceedings of the Workshop on ns-3*, pages 118–124. ACM, 2017.

[FCR18] Helder Fontes, Rui Campos, and Manuel Ricardo. Improving the ns-3 tracebased-propagationlossmodel to support multiple access wireless scenarios. In *Proceedings of the 10th Workshop on ns-3*, pages 77–83. ACM, 2018.

REFERENCES

[Feda] Fed4Fire+. https://www.fed4fire.eu/. Accessed: 2018-02-08.

[fedb] Fed4fire+ testbed sites image. https://www.fed4fire.eu/wp-content/uploads/sites/10/2017/06/testbeds.png. Accessed: 2018-02-08.

[FT00] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[Gena] Geni. http://www.geni.net/. Accessed: 2018-02-08.

[genb] Geni testbed sites image. http://groups.geni.net/geni/raw-attachment/wiki/Graphics/GENI-Map.png. Accessed: 2018-02-08.

[Gos] Brecht Gosselé. A survey on nosql and newsql datastores.

[HBa] HBase. https://hbase.apache.org/. Accessed: 2018-02-08.

[HR83] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*, 15(4):287–317, 1983.

[Jaz07] Mehdi Jazayeri. Some trends in web application development. In *2007 Future of Software Engineering*, pages 199–213. IEEE Computer Society, 2007.

[KAN+13] Shahbaz Khan, Bilal Aziz, Sundas Najeeb, Aziz Ahmed, Muhammad Usman, and Sadiq Ullah. Reliability of network simulators and simulation based research. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*, pages 180–185. IEEE, 2013.

[KLM+12] Stratos Keranidis, Wei Liu, Michael Mehari, Pieter Becue, Stefan Bouckaert, Ingrid Moerman, Thanasis Korakis, Iordanis Koutsopoulos, and Leandros Tassiulas. Concrete: A benchmarking framework to control and classify repeatable testbed experiments. In *FIRE Engineering Workshop*, 2012.

[Kno] KnockoutJS. http://knockoutjs.com/. Accessed: 2018-02-08.

[LCC+15] João Ricardo Lourenço, Bruno Cabral, Paulo Carreiro, Marco Vieira, and Jorge Bernardino. Choosing the right nosql database for the job: a quality attribute evaluation. *Journal of Big Data*, 2(1):18, 2015.

[Lea10] Neal Leavitt. Will nosql databases live up to their promise? *Computer*, 43(2), 2010.

[Mon] MongoDB. https://www.mongodb.com/. Accessed: 2018-02-08.

[Mur07] San Murugesan. Understanding web 2.0. *IT professional*, 9(4), 2007.

[ngi] Nginx. https://www.nginx.com/. Accessed: 2018-02-08.

[ns-] ns-3. https://www.nsnam.org/. Accessed: 2018-02-08.

[OL04] Philippe Owezarski and Nicolas Larrieu. A trace based method for realistic simulation. In *Communications, 2004 IEEE International Conference on*, volume 4, pages 2236–2239. IEEE, 2004.

[Ore10] Kai Orend. Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer. *Architecture*, page 100, 2010.

[Pri08] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008.

[Rea] ReactJS. https://reactjs.org/. Accessed: 2018-02-08.

[Rub] Ruby on Rails. http://rubyonrails.org/. Accessed: 2018-02-08.

[SA12] Aaron Schram and Kenneth M Anderson. Mysql to nosql: data modeling challenges in supporting scalability. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 191–202. ACM, 2012.

[Str11] Christof Strauch. Nosql databases. lecture selected topics on software-technology ultra-large scale sites. *Manuscript. Stuttgart Media University*, 2011.

[Sud03] Brian Suda. Soap web services. *Retrieved June*, 29:2010, 2003.

[thr] Apache Thrift. http://thrift.apache.org/. Accessed: 2018-06-22.

[ZP] Konstantinos Zarifis and Yannis Papakonstantinou. In-depth survey of mvvm web application frameworks.

REFERENCES

Appendix A

Captured metrics

This appendix presents an enumeration and description of the metrics that can be captured using this dissertation's information retrieval agent. It also shows the way that the databases used throughout the system are structured.
GPS data:

- expName: The experience identifier;
- nodeName: The node's identifier;
- pctimstamp: Timestamp representing when this data was written to the database;
- gpstimestamp: Timestamp recorded by the GPS device;
- longitude: Longitude recorded by the GPS device;
- latitude: Latitude recorded by the GPS device;
- altitude: Altitude recorded by the GPS device.

Info data:

- expName: The experience identifier;
- nodeName: The node's identifier;
- pctimstamp: Timestamp representing when this data was written to the database;
- noise: Noise recorded at the WLAN card;
- txpower: Transmission power recorded at the WLAN card.

Packet data:

- expName: The experience identifier;
- nodeName: The node's identifier;

Captured metrics

- timedate: Timestamp representing when this data was captured;
- wlantype: Type of message;
- srcMAC: MAC address of the packet's originating computer;
- dstMAC: MAC address of the packet's destination;
- bss: Associated base station;
- pkttype: Type of packet (for example, DATA or ACK)
- rssi: Received Signal Strength Indicator;
- idxrate: Index of transmission rate
- retry: Whether this packet was sent as the result of a retry attempt;
- pktlength: Length of the packet;
- rate: Data rate, in megabit/s * 10
- antenna1idx: Rate index of antenna 1
- rssi1: RSSI from antenna 1
- antenna2idx: Rate index of antenna 2
- rssi2: RSSI from antenna 2
- antenna3idx: Rate index of antenna 3
- rssi3: RSSI from antenna 3
- essid: Extended Service Set Identification
- mode: Trnasmission mode, related ti the packet type
- freq: Frequency at which the packet was transmitted;
- channel: Channel in which the packet was transmitted;
- seqnr: TCP Sequence number on the packet;
- srcip: IP address of the packet's originating computer;
- dstip: IP address of the packet's originating computer;
- loctimestamp: Unix timestamp;

Appendix B

Hardware specifications

B.1 Alix nodes

- CPU: AMD Geode x86;
- RAM: 256MB;
- Storage: Universal Serial Bus (USB) 2.0 - 8GB;
- Operating System: L.E.D.E / OpenWRT 17.01;
- Wi-Fi card: R52;
- GPS receiver.

B.2 Computer used for testing

- CPU: Intel i7-4720HQ;
- RAM: 8GB;
- Storage: Hard Drive Disk (HDD) - 750GB;
- Operating System: Arch Linux.

Appendix C

Questionnaire

The following figures represent the questionnaire which was distributed as a method of validating this dissertation's proposed solution's impact.

Usability inquiry - New framework for automating the recording and reproduction of wireless experiments in ns-3

In [1] Fontes et al. introduced a new approach enabling repeatable and reproducible wireless experiments. This approach explores how real traces of Signal-to-Noise Ratio (SNR) and node positions can be used to reproduce past wireless experiments conditions in ns-3 via trace-based simulations.

The objective of this framework is to automate (1) process of creating real traces from wireless experiments; and (2) the process of creating trace-based simulations using those real traces to reproduce those same experiments.

This inquiry was created to assess the impact of this framework regarding the necessary work methodology a network researcher needs to follow to successfully use the trace-based simulation approach to create repeatable and reproducible experiments.

Please consider the necessary steps to produce a trace-based simulation without using this framework:

1. Developing a specialized script/program to gather the necessary data (real traces, network configurations, ...);
2. Access each node manually to retrieve said data;
3. Manually organize the data, and edit where necessary;
4. Format the data so that it can be used in an ns-3 trace-based simulation;
5. Manually create the ns-3 scenario that uses the data.

Note also that at all of these stages, there is a non-negligible chance for human error to occur, which can potentially taint the experiment's data.

Now, please consider the necessary steps to produce a trace-based ns-3 simulation using this framework:

1* Downloading the information retrieval agent onto each node;
2* Configuring and running the program on each node;
3* Going to the website and downloading the automatically generated and pre-configured trace-based ns-3 simulation file with the real respective traces.

[1] Helder Fontes, Rui Campos, and Manuel Ricardo. 2017. A Trace-based ns-3 Simulation Approach for Perpetuating Real-World Experiments. In Proceedings of the Workshop on ns-3 (WNS3 '17). ACM, New York, NY, USA, 118-124. DOI: https://doi.org/10.1145/3067665.3067681

Figure C.1: The questionnaire's preliminary exposition.

How long do you think it would take you to go through steps 1-5 (process WITHOUT the framework)? *

Choose

How long do you think it would take you to go through steps 1*-3* (process WITH the framework)? *

Choose

What would you say is the chance that, throughout the process WITHOUT the framework, some sort of human error occurs? *

	1	2	3	4	5	
Impossible	○	○	○	○	○	It will happen for sure

In the process WITHOUT the framework, where do you think that human error is most likely to occur and why? *

Your answer

What would you say is the chance that, throughout the process WITH the framework, some sort of human error occurs? *

	1	2	3	4	5	
Impossible	○	○	○	○	○	It will happen for sure

In the process WITH the framework, where do you think that human error is most likely to occur and why? *

Your answer

Would you use this framework? *

○ Yes

○ No

Figure C.2: The questionnaire's questions.

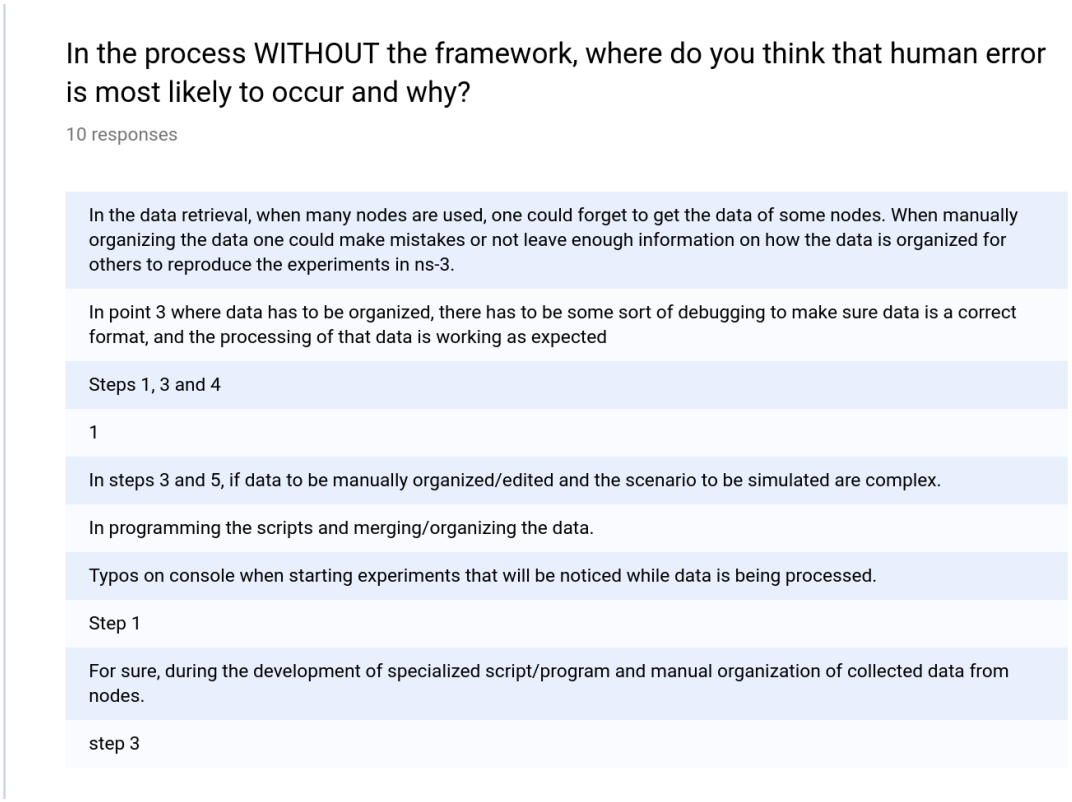

Figure C.3: The questionnaire's questions.

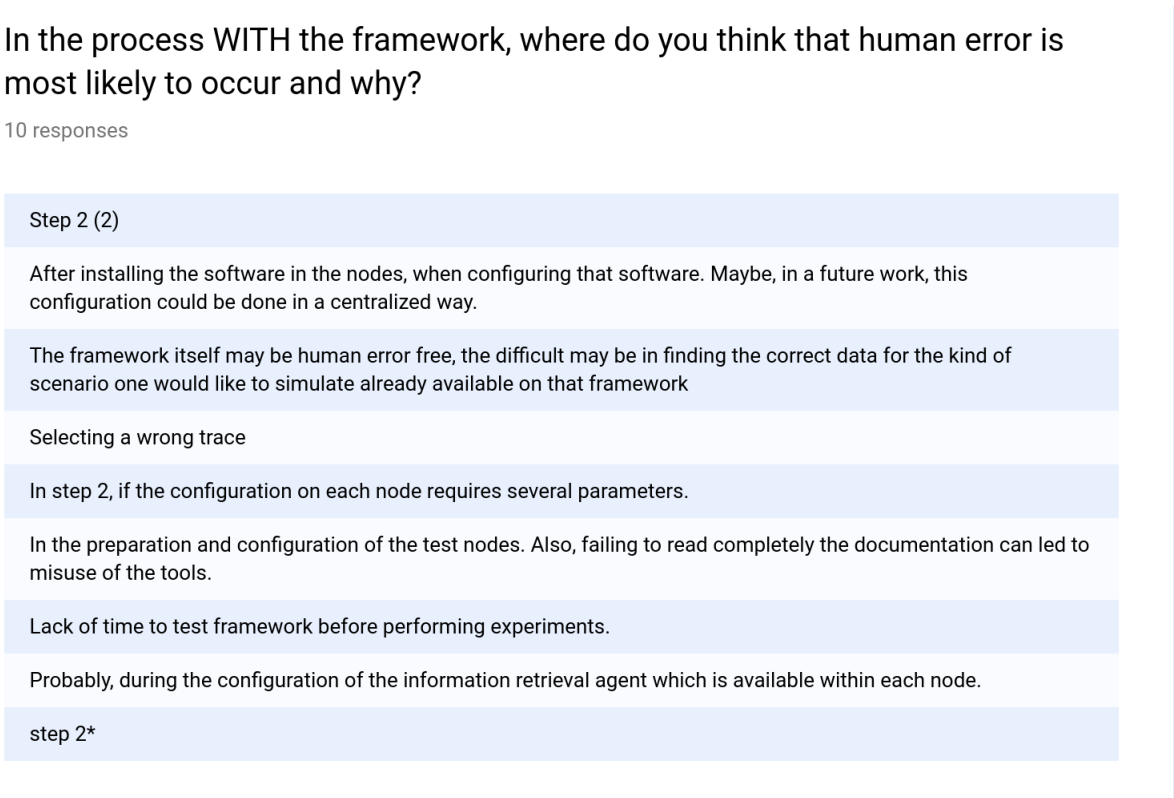

Figure C.4: The questionnaire's questions.

Appendix D

Guidelines for future improvements

D.1 Information Retrieval Agent

To further increase the modularity of the agent, the data capturing modules the user desires to use should be defined in the configuration file. This will allow for the launcher module to dynamically decide which launchers to use without the user having to modify the code to introduce or remove data capturing modules.

Even though this component was developed with modularity and expandability in mind, the looseness of its modules can still be improved, so that each module is fully replaceable. For this, the agent could be refactored into a more object oriented architecture, leveraging classes, polymorphism, and inheritance.

In this particular use case, the only relevant packets are the ones that are sent or received by the nodes involved in the experiment. To that end, a MAC filtering mechanism was developed, which makes it so that only packets that have one of the experiment's nodes' MAC address either as the destination or the source are stored. However, this feature makes the user specify explicitly which MAC addresses are involved in the experiment, which is time consuming and can also lead to human error. To mitigate this problem, IP filtering could be implemented, which would allow for wildcards. This would be helpful since the experiment's nodes are usually all in the same network, and as such the relevant nodes can be easily declared with a generic expression that matches the network prefix.

The hardware that this agent was tested on remained the same throughout the whole development process. Although the agent's performance is acceptable when run in that hardware, since the agent is meant to perform on multiple different devices and seeing as most network devices have relatively low performance hardware, the agent's performance should be optimized. This can be achieved either by doing a thorough profiling and analysis of the code and optimizing where possible or by rewriting the agent in a more performance-oriented language, such as C++. This would also work in terms of moving toward a more object-oriented architecture, as mentioned above. The disadvantage of moving to a compiled language such as C++ is that new issues would arise,

such as the need to cross-compile the agent and potentially alter the implementation according to the architecture it will run on.

Having to generate a configuration for each node in an experiment can be dull, even if the process involves simply doing one configuration and editing it for each node. To tackle that problem, an improved setup script could be developed which would automatically generate configuration and database files for every node in the experiment, with the option to also transmit those files to the node automatically.

Furthermore, in the current solution, the agent has to be manually started on each node, which might be a hassle. This issue can be mitigated with implementation of an additional component on the solution, which would remotely control the agent. This component would establish a communication channel with each node and exchange control messages, such as messages to start, pause and end the capturing process.

This new component could also be availed to tackle the problem of manually generating configurations for each node and even to transfer the data, although this last option might prove to be too cumbersome on the network. If the machine running this controller were to be within the experiment's network, for example in a proxy node, communication could even be performed via multicast.

Also, the logs produced by these nodes are kept in the node's storage, so that if any events occur that would affect the program's performance, a record could be kept for debugging purposes. However, it would be simpler to have the aforementioned controller node be able to toggle the logging of each node as well as receive notifications when any event occurs that requires a user's attention.

Lastly, the metrics that are collected by this agent are hard-coded. A way to improve this would be to have the controller node decide which metrics are collected. Also, both the temporary databases within the nodes could be flexible enough that if any new table is required, it could be built at runtime and without interrupting the gathering of data.

D.2 REST API and Back-end

One of the most constraining issues with this solution performance-wise is that a request for data of a certain kind will return all of the data in one request. This makes the message big enough that its transmission may take an unacceptable amount of time. To solve this, the endpoints should support the option of returning only a selected number of entries for the requested data type.

As mentioned in Section D.1, the system should be able to adapt to the type of data the user wants to collect. To that end, the central database should also be able to adapt by building new data tables at runtime, if necessary, as should the API endpoints.

Furthermore, a new endpoint could be developed which would run the trace-based simulations on the web server. This endpoint could then be integrated into the web interface in order to produce real time simulation output and its results via WebSockets. This feature would help to further mitigate the requirement for specialized knowledge involved in network simulation.

Moving to HTTPS would provide safer message exchange, since this protocol includes encryption. Furthermore, browsers nowadays are starting to disallow HTTP content, so this transition would help keep the system usable and up-to-date.

The fact that the API has no sort of restriction at this time is also a security issue, since it may lead to malicious injection of data or to external users unlawfully obtaining data. There are a number of approaches to tackle this. The first would be to restrict API access to some select IP addresses, thus guaranteeing that the origin of a request is known. This would, however, hinder global access to the data. Another, more reasonable, approach would be to create user accounts and restrict API access to logged in users.

Creating a user system would enable some additional features, such as having multiple user groups. This would be useful for further detailing API access permissions. For example, a user group could be created for information retrieval agents with permissions to access only the POST method of the */data* endpoint, while the regular user group would only have GET access to the three individual data endpoints.

Coupled with the creation of user groups, new endpoints could be developed for the administrator user group which would grant PATCH access to the individual data streams' endpoints in order to allow for the editing of data. Administrators could also approve data before it became public, thus ensuring its credibility and correctness. For this, the database would have to be split into tables with approved and raw data. GET requests to the endpoints would return only approved data and new endpoints, again, only available to administrators, would return raw data.

Finally, with a user system in place, it would be possible to have a user claim ownership of data. This can be done by associating certain information retrieval agents' accounts to a certain user, which can be achieved by having these agents send a pre-generated identification token in the headers of their HTTP requests. This would then enable the development of new endpoints which are able to filter data by user. Moreover, users could decide whether they want their data to be public or not, thus creating the notion of privacy in the platform. For a more fine grained control of data access, it could even be possible to have a user whitelist access to some of his data.

D.3 Web interface and Front-end

The one page that currently composes this solution's web interface exhibits a number of problems. One is the fact the packet data table, being 26 columns long, goes beyond the bounds of the screen, making it necessary to scroll the page to observe some of its metrics. Furthermore, when the table is scrolled, it eventually overlaps with the command module on the left side of the screen, potentially making it impossible to access the buttons and input field that comprise it. This can be solved by defining the area containing the data table as not being able to shift onto the command module using CSS, so that, when scrolling the table, it will only scroll the right side of the screen. Alternatively, it would also be possible to create some sort of controls which would cycle the values in the table, so that no scrolling would be necessary. Instead, when pressing a button, all columns would shift right or left by one, thus revealing the previously unseen metrics.

In alternative to loading all of the data at once, the front-end application should lazy-load the data. Lazy-loading is the process of only loading data when it is expressly necessary. In this interface, this means only requesting data concerning the current page to the backend API. When going to a new page, new data is downloaded and stored so that if the user switches to a new page and then back, a new request is not necessary. In this way, the initial waiting time for all of the data to load is eliminated.

When downloading either the CSV file containing the currently available data or the simulation files, the data set that is used is the one that is currently loaded in the application. This might cause problems if the data had some sort of filter applied at the time when the files were generated. Also, the interface fails to inform the user of what it is doing and which data is being included in the file generation. To tackle these issues, some approaches are suggested: First, the application should be more interactive and informative, so that users can navigate it without any problems or the need for specialized knowledge. Second, the interface should notify the user and ask for its confirmation when a file generation is requested while the data set has filters applied.

At some point, access to this platform will be open to a larger audience. When this happens, it would be useful and also more appealing and professional to have a landing page as well as a set of pages which provide information about the platform, what it is for and how it works.

Given the suggestions for improvement made in Section D.2, the web application should also provide a way for users to authenticate themselves on the platform in order to be able to manipulate data according to the permissions granted to them by the system.

It would also be necessary to develop new pages in the application, such as a page where system administrators could access and approve raw data and a page for the currently authenticated user to interact with its own data.

Finally, a new page could be developed where a user can receive real-time output from the reproduction of a given experiment that would be running server-side. This page would also allow for the download of the simulation's results.